Scheduling Streaming Operators for IoT Edge Analytics

Patient Ntumba wa Ntumba

Sorbonne Université

École Doctorale Informatique, Télécommunications et Électronique (Paris)

Inria Paris / MiMove Team

Scheduling Streaming Operators for IoT Edge Analytics

By **Patient Ntumba Wa Ntumba**

Doctoral thesis in Computer Science

Under the supervision of Nikolaos Georgantas and Vassilis Christophides

Presented and defended publicly on September 9th, 2022

The jury is composed of:

Prof. Bernd AMANN (Sorbonne Université, FR)	President
Prof. Panos KYPROS CHRYSANTHIS (University of Pittsburgh, USA)	Reviewer
Dr. Cédric TEDESCHI (Université de Rennes 1 & INRIA, FR)	Reviewer
Prof. Frédéric LE MOUËL (INSA Lyon, FR)	Examiner
Prof. Patricia STOLF (Université de Toulouse Jean Jaurès & IRIT, FR)	Examiner
Dr. Nikolaos GEORGANTAS (Inria, FR)	Advisor
Prof. Vassilis CHRISTOPHIDES (ENSEA & ETIS, FR)	Co-advisor

Abstract

Data stream processing and analytics (DSPA) applications are widely used to process the ever increasing amounts of data streams produced by highly geographically distributed data sources, such as fixed and mobile IoT devices, in order to extract valuable information in a timely manner for actuation. DSPA applications are typically deployed in the Cloud to benefit from practically unlimited computational resources on demand.

However, such centralized and distant computing solutions may suffer from limited network bandwidth and high network delay. Additionally, data propagation to the Cloud may compromise the privacy of sensitive data.

To effectively handle this volume of data streams, the emerging Edge/Fog computing paradigm is used as the middle-tier between the Cloud and the IoT devices to process data streams closer to their sources and to reduce the network resource usage and network delay to reach the Cloud. However, Edge/Fog computing comes with limited computational resource capacities and requires deciding which part of the DSPA application should be performed in the Edge/Fog layers while satisfying the application response time constraint for timely actuation. Furthermore, the computational and network resources across the Edge-Fog-Cloud architecture can be shareable among multiple DSPA (and other) applications, which calls for efficient resource usage.

In this PhD research, we propose a new model for assessing the usage cost of resources across the Edge-Fog-Cloud architecture. Our model addresses both computational and network resources and enables dealing with the trade-offs that are inherent to their joint usage. It precisely characterizes the usage cost of resources by distinguishing between abundant and constrained resources as well as by considering their dynamic availability, hence covering both resources dedicated to a single DSPA application and shareable resources. We complement our system modeling with a response time model for DSPA applications that takes into account their windowing characteristics.

Leveraging these models, we formulate the problem of scheduling streaming operators over a hierarchical Edge-Fog-Cloud resource architecture. Our target problem presents two distinctive features. First, it aims at jointly optimizing the resource usage cost for computational and network resources, while few existing approaches have taken computational resources into account in their optimization goals. More precisely, our aim is to schedule a DSPA application in a way that it uses available resources in the most efficient manner. This enables saving valuable resources for other DSPA (and non DSPA) applications that share the same resource architecture. Second, it is subject to a response time constraint, while few works have dealt with such a constraint; most approaches for scheduling time-critical (DSPA) applications include the response time in their optimization goals.

To solve our formulated problem, we introduce several heuristic algorithms that deal with different versions of the problem: static resource-aware scheduling that each time calculates a

new system deployment from the outset, time-aware and resource-aware scheduling, dynamic scheduling that takes into account the current deployment.

Finally, we extensively and comparatively evaluate our algorithms with realistic simulations against several baselines that either we introduce or that originate / are inspired from the existing literature. Our results demonstrate that our solutions advance the current state of the art in scheduling DSPA applications.

Keywords

Internet of things, data stream, Continuous Operator, Edge/Fog computing, Cloud Computing, Optimization, Queueing Networks, Dynamic System

Résumé

Les applications de traitement et d'analyse des flux de données (DSPA en anglais) sont largement utilisées pour traiter les quantités toujours plus importantes de flux de données produites par des sources de données hautement distribuées géographiquement, telles que les dispositifs de l'internet des objets (IdO) fixes et mobiles, afin d'extraire des informations précieuses le plus rapidement possible pour une action satisfaisaint une limite de temps de réponse. Les applications DSPA sont généralement déployées dans le Cloud pour bénéficier de ressources de calcul pratiquement illimitées à la demande. Cependant, ces solutions de calcul centralisées et distantes peuvent souffrir d'une bande passante réseau limitée et des retards de réseau élevé. De plus, la propagation des données dans le nuage peut compromettre la confidentialité des données sensibles.

Pour traiter efficacement ce volume de flux de données, le paradigme émergent du Edge/Fog computing est utilisé comme niveau intermédiaire entre le Cloud et les dispositifs IdO pour traiter les flux de données plus près de leurs sources afin de réduire l'utilisation des ressources réseau et les retards dans le réseau pour atteindre le Cloud. Cependant, le paradigme Edge/Fog computing contient des ressources de calcul limitées, il est donc necessaire de décider quelle partie de l'application DSPA doit être exécutée au niveau du Edge/Fog tout en satisfaisant à la contrainte de temps de réponse de l'application. De plus, les ressources de calcul et de réseau de l'architecture Edge-Fog-Cloud peuvent être partagées entre plusieurs applications de DSPA (ou autres), ce qui nécessite une utilisation efficiente de ces ressources.

Dans cette thèse, nous proposons un nouveau modèle pour évaluer le coût d'utilisation des ressources à travers l'architecture Edge-Fog-Cloud. Notre modèle concerne à la fois les ressources de calcul et de réseau et permet de traiter les compromis inhérents à leur utilisation conjointe. Ce modèle caractérise précisément le coût d'utilisation des ressources en distinguant les ressources abondantes des ressources contraintes et en considérant leur disponibilité dynamique, couvrant ainsi les ressources dédiées à une seule application de DSPA et les ressources partageables. Nous complétons notre modélisation du système par un modèle de temps de réponse pour les applications DSPA qui prend en compte leurs caractéristiques de fenêtrage.

En s'appuyant sur ces modèles, nous formulons le problème de l'ordonnancement d'opérateurs continus, qui constituent une application de DSPA, sur une architecture hiérarchique de ressources Edge-Fog-Cloud. Notre problème cible présente deux différentes caractéristiques. Premièrement, il vise à optimiser conjointement le coût d'utilisation des ressources de calcul et de réseau, alors que peu d'approches existantes ont pris en compte les ressources de calcul dans leurs objectifs d'optimisation. Plus précisément, notre objectif est de déployer une application de DSPA de manière à ce qu'elle utilise les ressources disponibles de la manière la plus efficace possible. Cela permet d'économiser des ressources précieuses pour les autres applications de DSPA (ou d'autre type) qui partagent la même architecture de ressources.

Deuxièmement, il est soumis à une contrainte de temps réponse, alors que peu de travaux ont traité d'une telle contrainte ; la plupart des approches d'ordonnancement des applications soumises au contrainte de temps de réponse incluent le temps de réponse dans leurs objectifs d'optimisation.

Nous introduisons plusieurs algorithmes basés sur des heuristiques qui traitent différentes versions du problème : l'ordonnancement statique tenant compte que des ressources de calcul et réseau, l'ordonnancement static tenant compte à la fois des ressources et de la contrainte de temps de réponse, et l'ordonnancement dynamique qui prend en compte le déploiement actuel de l'application et des resources disponibles.

Enfin, nous évaluons de manière approfondie et comparative nos algorithmes à l'aide de simulations réalistes par rapport à plusieurs alogorithmes que nous avons soit conçu ou qui sont issus ou inspirés de la littérature existante. Nos résultats démontrent que nos solutions font progresser l'état actuel de l'art en matière d'ordonnancement des applications de DSPA.

Mots clés

Internet des objets, flux de données, opérateur continu, informatique de périphérie/de brouillard, informatique en nuage, optimisation, réseaux de files d'attente, système dynamique

Quote

To my beloved parents

Acknowledgements

The last four years that I spent on my thesis were not only an intellectual challenge for me but also one of self-improvement, during which I learned how to conduct a research project in a systematic way. It was not only thanks to my effort but also the support of many people that I was able to achieve this outcome.

This work would not have been possible without the constant support, guidance, and assistance of my Ph.D advisor Dr. Nikolaos Georgantas and my Ph.D. co-advisor Pr. Vassilis Christophides. Their levels of patience, knowledge, and ingenuity is something I will always keep aspiring to. I would like to thanks the members of the jury committee of my thesis, specifically to Pr. Panos Kypros Chrysanthis and Dr. Cédric Tedeschi for accepting to serve as the reviewers and for their invaluable suggestions and feedback, to Pr. Bernd Amann for accepting to serve as the president of the jury and to Pr. Patricia Stolf and Pr. Frédéric Le Mouël for accepting to serve as the examiners.

Many thanks to all of the members of the MiMove team staff for their kind support during my PhD study, specifically Nathalie Gaudechoux, Valérie Issarny, Françoise Sailhan, Maroua Bahri, Abdoul Shahin Abdoul Soukour, and William Aboucaya. Also, I extend my thanks to the former members of the MiMove team Georgios Bouloukakis, Yifan Du and Rafael Angarita for their continuous encouragement and support in the earlier stage of my PhD study.

Finally, I express my deepest gratitude to my wife Marianne for her support and love during these four years and to my familly and wife's family for their incentives. I would also like to thanks my friend Vianney Lotoy Bendenge who has accompanied me since the beginning.

Table of contents

Chapter 1

Introduction

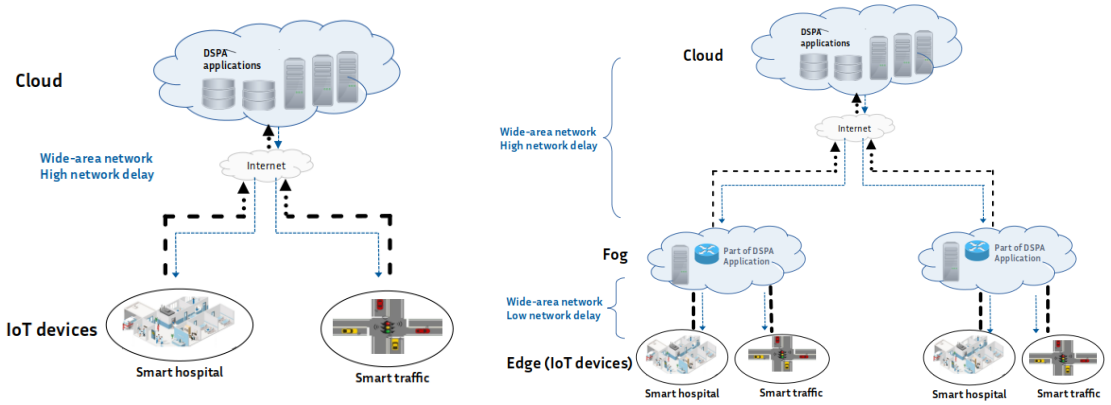
Contents

This chapter contains 11 pages.

Today, more and more data are delivered in real-time. According to the International Data Corporation[1] by 2025, more than 25% of all data created will be real-time in nature [2]. *Real-time data* could not be stored but should be processed as quickly as it is gathered. The *time value* of data is essential in many applications with time-constraint control and automation, such as smart transportation [1, 2], augmented or virtual reality [3–6]. For example, Internet of things (IoT) applications built using several geographically distributed devices (aka thinks) enhanced with sensing capabilities produce data streams that have to be processed and analysed on the fly in order to take time-sensitive decisions or actions e.g., to regulate a traffic jam. Unlike snapshot queries on historical data, *data stream processing and analytics* (DSPA) engines continuously process unbounded data streams to extract valuable information in timely manner via a series of *continuous operators (streaming operators or simply operators)* such as aggregation, filter, join, etc [7]. This series of operators specifies a DSPA application.

DSPA applications are typically deployed in the Cloud to benefit from unlimited computational resources on demand depending on the number of streams and the speed of data that needs to be processed. As depicted in Figure 1.1a, this Cloud-based analytics requires to transmit all the data streams produced by IoT devices to the distant Cloud by traversing wide area network (WAN) links involving several hops [8] and the response may have to be sent all the

[1] https://www.idc.com/
[2] https://www.zdnet.com/article/by-2025-nearly-30-percent-of-data-generated-will-be-real-time-idc-says/



(a) Cloud IoT Analytics architecture

(b) IoT edge Analytics architecture

Figure 1.1: From Cloud based IoT analytics to IoT edge analytics

way back. Such centralized solutions favor DSPA application availability but may suffer from network congestion and delay issues in case of highly dynamic data stream rates. Additionally, data propagation to the Cloud may compromise privacy of sensitive data [9].

Recent breakthroughs in network technology allows to realize post-Cloud architectures in order to collect and process real-time data. In particular, 5G networks [10,11] enable an increased network bandwidth capacity up to 10Gbps, low-latency communications down to 1ms, and a high connectivity density up to 1 million of IoT devices per km^2. This enables IoT devices to transmit a greater number of data streams at high data rates. However, pushing systematically to the Cloud such amount of data streams could over-utilize the available network resources and thus introduce network delays. The situation is worsened by the fact that the available bandwidth capacities of WAN links are inherently dynamic due to the varying Internet traffic conditions [8]. Additionally, the sum rate of transmissions of IoT data streams could overwhelm the bandwidth capacity reserved in the Cloud, incurring additional monetary charges: the cost for WAN bandwidth usage can be much higher than the cost for computational resource usage [12].

In this respect, bringing computational resources closer to where data streams are produced (at the IoT network edge) through Edge and Fog architectures is currently one of the key solutions being adopted by the industry to respond to the identified need, but also an open research question [13, 14]. As depicted in Figure 1.1b, the Edge/Fog architecture introduces intermediate layers of computing, storage, and networking between the IoT devices and the Cloud enabling to process data streams near their sources and hence to reduce network consumption and network delay of WAN links toward the Cloud and to enforce data privacy [15, 16].

Edge/Fog architectures are highly distributed and come with stringent resource constraints in terms of computational capacity (e.g., CPU, GPU, RAM, etc.) or power supply (e.g., recharge-

able batteries, solar energy, wind power,etc.). In this context, the allocation of Edge/Fog computational resources to a DSPA application should cope with their *heterogeneity*, their *limited* and *shareable capacities*, as well as the *highly dynamic workload* related to the spatio-temporal dynamics of real-time data produced in the wild [17, 18].

In order to process as much as possible data streams at the IoT network edge, one needs to schedule the related operators across Edge-Fog-Cloud nodes by controlling both the processing latency and the network delays, while keeping minimal the costs of using the computational resources (CPU/GPU, RAM, energy) at the Edge/Fog layers and the network resources to reach the Cloud (WAN bandwidth, WAN delays) [14]. Specifically, we aim to schedule the operators across Edge/Fog/Cloud nodes in a way that exhibits *optimal trade-offs* between the usage of *network and computational resources*. As DSPA application workload may highly vary, we need to monitor at run-time the allocated resources and *continuously reschedule* the operators in the Edge/Fog/Cloud continuum.

Scheduling operators has largely been studied in the literature for DSPA applications deployed in the Cloud or distributed on peer nodes of a data center, where the computational resources are abundant [19, 20]. On the other hand, related work on processing data streams at the IoT network edge mostly focuses on reducing the required network bandwidth and resulting delays to reach the Cloud by exploiting to the maximum the available computational resources at the Edge/Fog layers [21–24]. Given that the Edge/Fog nodes come with heterogeneous and limited computational resources, using to the maximum their available computational resources may in turn impair on the DSPA application performance i.e. violate time-constraint [14].

1.1 Use case: Wide-area traffic management

To motivate this thesis, we introduce a DSPA application for wide-area traffic management [25] at different geographical and time scales as illustrated in Figure 1.2. We consider streams of traffic data generated by connected vehicles (simply vehicles) such as the vehicle identifier, GPS-location and driving speed. Vehicles transmit such data at a given frequency to the nearest street antenna which in turn forwards them to the DSPA application.

The DSPA application processes the input traffic data by the means of operators. Given the infinite nature of IoT data stream, we consider for each operator a window to chop its input data streams into a finite set of data to be processed along a fixed time period (window size).

The resulting data stream after being processed by the DSPA application via the series of operators is given as input for two IoT analytics applications. The first implements a *country-wide traffic monitoring* that reports on an hourly basis traffic statistics for the entire country. The second application aims to support *city-wide traffic regulation*: control the traffic lights, e.g. to give priority to jammed traffic flows over non-jammed ones, etc.

We assume that the country-wide traffic monitoring application is deployed in the Cloud

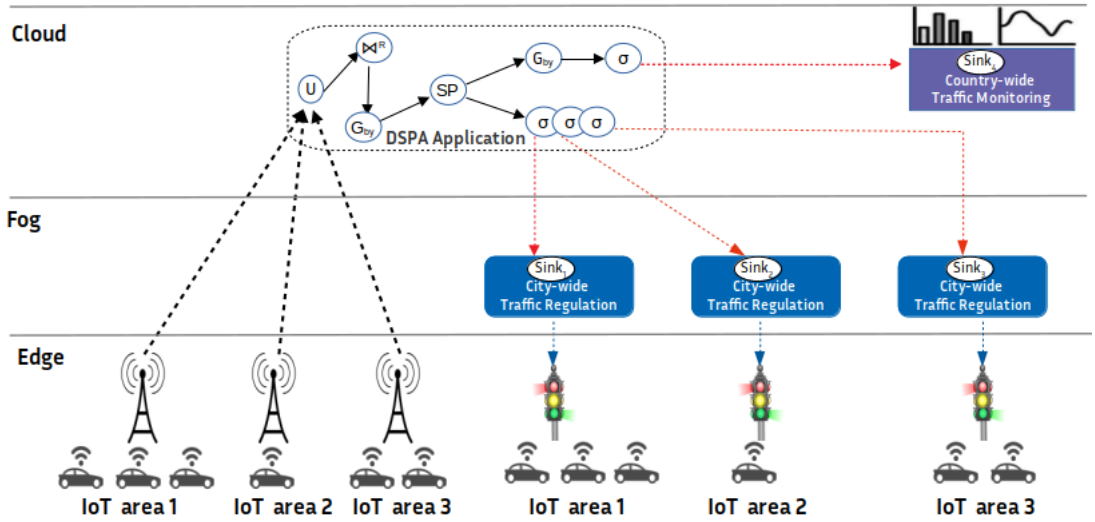


Figure 1.2: DSPA application for wide-area traffic management

and an instance of city-wide traffic regulations is deployed per district on the Fog to regulate the traffic per IoT area. Furthermore, we assume that the DSPA application is deployed in the Cloud and the available Fog computational resource capacity can be used by other applications including also the DSPA application if necessary.

Traffic data rates are highly dynamic due to the high variability of traffic density and mobility patterns of vehicles [26] between rush hours (high data stream rates) and normal hours (normal data stream rates). This can lead to high fluctuations in the DSPA's demand for processing capacity and network bandwidth, hence may result in resource congestion and failure to meet time-constraint requirements. In particular, at normal hours DSPA application can be deployed in the Cloud as the data streams send to the Cloud at normal rate do not overwhelm the WAN resources and the aggregated data stream rates do not exceed the maximum Cloud bandwidth capacity agreed between the Cloud provider and the application owner. However, at rush hours sending data streams at high rates toward the Cloud may overwhelm the WAN resources causing network congestion and high network delays that may impair on the timeliness of data stream. Furthermore, the aggregated data stream rates may eventually exceed the maximum Cloud bandwidth capacity and therefore unforeseen monetary costs.

Pushing computation of DSPA application at the IoT network edge should be exploited to overcome high variability in the number or the rate of traffic data streams. In this case data streams will be partially processed at the IoT network edge and the reduced data stream rates will be sent to the Cloud for further processing. Of course the computational resources requested by the pushed computations should not exceed the capacity of the IoT network edge on which a part of the DSPA application is deployed. Furthermore, when placing a part of DSPA application

at the IoT network edge, one needs to ensure *geographical placement constraint*. In particular, each part of DSPA application can be placed at specific node at the IoT network edge if there is sense in applying its operation only to the local data stream produced by the closest IoT area (city level). Otherwise, if this operation must be applied to the global data stream (country level), this part can only be deployed in the Cloud.

1.2 Scope of the thesis

This thesis considers a hierarchical Edge-Fog-Cloud architecture that can be shared among several DSPA applications. Each IoT device at the Edge is connected to its closest Fog node and all the Fog nodes are connected to a single Cloud. According to the use case example, we assume that a DSPA application is initially deployed in the Cloud, on which the sink of the DSPA application is fixed, while some part of the DSPA application can migrate to the Fog if necessary. Figure 1.3 depicts the hierarchical Edge-Fog-Cloud architecture in consideration. At the root of the hierarchy we consider a Cloud node providing computational and network resources on demand. Linked to the Cloud node, a set of Fog nodes provides heterogeneous and limited computational and network resources at the Fog layer [27]. We assume an overlay network based on the publish-subscribe protocol MQTT [28] to transport data streams across the Edge-Fog-Cloud architecture. In this respect we consider at the Edge layer, mobile IoT devices (e.g.,vehicles) per geographical area that produce and publish data at a certain rate (e.g., 4KB/s [29]) to the MQTT instance deployed on the closest Fog node via 5G antennas. We assume that these antennas enable high connectivity density and lossless communication for IoT devices. If a part of the DSPA application is deployed on this Fog node, the MQTT instance transfers these data to this application segment and waits for the resulting output data back in order to send them to the MQTT instance deployed in the Cloud for further processing. Otherwise, if no application part is deployed on the Fog node, the Fog MQTT instance transfers directly these data to the Cloud MQTT instance.

In this setting, we are interested in continuously scheduling DSPA application between the Fog and Cloud nodes for processing the data streams generated by IoT devices at the Edge. In particular as data stream rates may evolve according to various spatio-temporal patterns [17, 18], the DSPA application workload may heavily vary, and accordingly the computational and network resources needed to be allocated for its processing. Taking for instance the use case example in Section 1.1, in case of high usage of computational and network resources, we essentially need to reschedule the current placement of operators by replicating new operators initially deployed at the Cloud or removing already replicated operators at the Fog. To this, we need to address the following challenges:

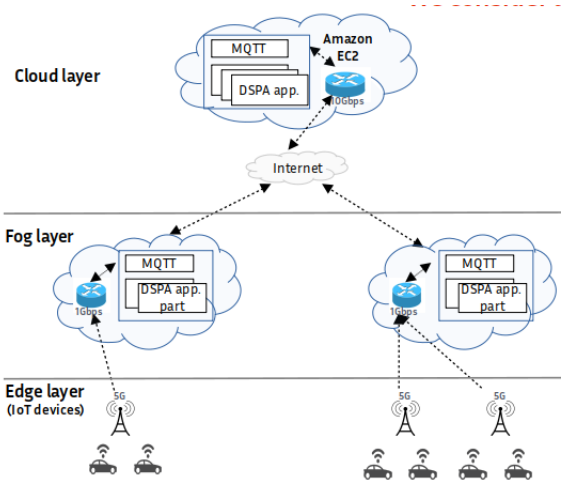


Figure 1.3: Edge-Fog-Cloud architecture example

How to model the resource usage of DSPA applications across a shareable and heterogeneous Edge-Fog-Cloud architecture ? Initially modeling computational and network resources across the Edge-Fog-Cloud architecture is challenging due to the heterogeneity of the different nodes in terms of CPU/RAM and network bandwidth capacities, as well as to the fact that this infrastructure is shared among several (DSPA or other) applications. As a matter of fact, several stakeholders (e.g., the city, telecommunication companies, etc.) may provide their own network communication technology (e.g., WiFi, LoRaWAN, 4G, 5G, etc.) and equipment for Edge/Fog nodes (e.g., small server, Raspberry Pi, Gateway, 5G antennas, etc.). Furthermore, the geographic dispersion of Edge/Fog nodes with respect to the Cloud may increase the variability in terms of network delays.

On the other hand, DSPA applications are characterized by different type of operators (e.g., aggregation, filter, join, etc.) and each operator may support different type of windows (e.g., count, time-based). By assuming that a DSPA application processes data streams that come at different rates, it is necessary to model at fine grain each operator and links between operators in order to identify at any time instant its processing time and its computational and network resource usage demands for processing its input data stream and for transmitting the processed data streams to its upstream operator (or sink).

In this respect, given the heterogeneity of the computational and network resources across the Edge-Fog-Cloud architecture. It is necessary to distinguish these resources in terms of their maximum capacities. On the other hand, given that several (DSPA) applications can be deployed dynamically on the fly on these resources, it is also necessary to distinguish these resources in terms of their available capacities. It should also take into account the time-constraint of DSPA application. Therefore, we need to propose a holistic resource usage model as well as response time model that captures the different characteristics of both the Edge-Fog-Cloud architecture,

the DSPA application, and dynamic data stream stream rates produced by IoT devices.

How to deploy a DSPA application while taking into account the Edge-Fog-Cloud resource characteristics? Given that data streams produced by each individual IoT areas at the Edge reach the Cloud through the intermediate closest Fog nodes. Then, by assuming that all the operators that constitute a DSPA applications are usually deployed in the Cloud, distributing this application between the Cloud and Fog nodes assumes placing some operators on the Fog nodes. However only placing the existing operators of the DSPA application will not enable to handle all the data streams produced at the different IoT areas and the analytic results may not be accurate. Thus, we need to replicate each operator at the maximum on each of the Fog nodes on which the data stream arrive first before reaching the Cloud in order to partially processed these streams at the Fog layer.

However, given the different type of operators that determine the computational and network resource demands for each operator to process its input data stream and to send the processed data stream to next operator. We need to identify how to replicate these operators in respect of not only the maximum capacities but also the available capacities of Fog nodes capable of hosting them and of the Fog to Cloud network resources. Furthermore, we should pay attention whether some DSPA application may be subject to geographical placement constraint.

How to ensure that a deployed DSPA application satisfies the response time constraint?

The response time of a DSPA application is constituted by the operator processing time that depends on the characteristics of the operators, the available computational and network resources across the heterogeneous Edge-Fog-Cloud architecture, as well as the network delays in the network links through which operators transmit (or receive) data to each other (or from IoT devices). Any deployment of DSPA application does not necessarily guarantee to satisfy the response time constraint. Thus, we need to introduce a scheduling solution for DSPA applications across the Edge-Fog-Cloud architecture that optimizes not only the resource usage and satisfies the geographical placement constraint but also satisfies the response time constraint.

How to continuously ensure response time constraint and optimized resource usage of a DSPA application in dynamic environments? As discussed earlier, the workload of a DSPA application may exhibit variation at run-time due to variability of the rates of data streams produced by IoT devices. At the same time the available resource capacities across the Edge-Fog-Cloud architecture may exhibit also variation given these resources can be shared among several (DSPA) applications. Finally, the WAN resources to reach the Cloud can be dynamic due to the underlying internet condition [8]. Considering this dynamic environment on which a DSPA application can be deployed, the initial deployment, at a certain time may no longer optimize the resource usage or may fail to satisfy the response time constraint. Therefore, we

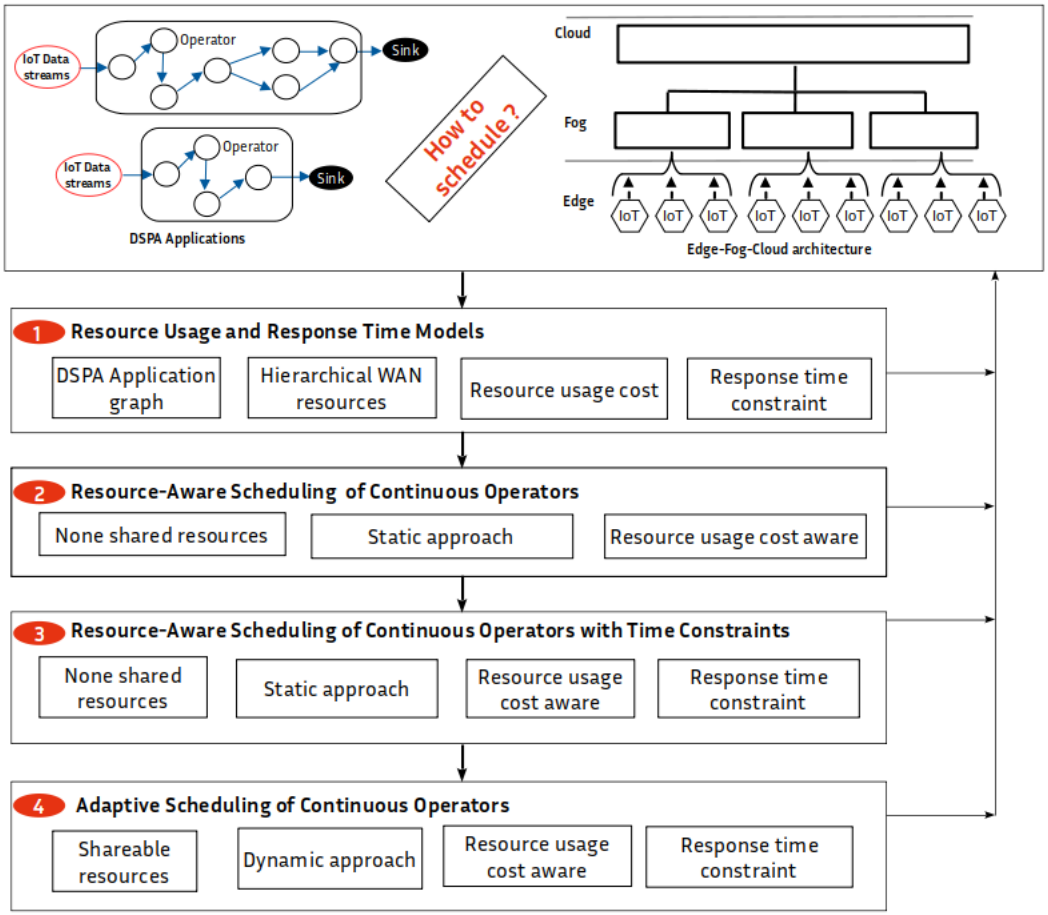


Figure 1.4: Framework for Scheduling of Continuous Operators for IoT edge analytics

need to monitor at run-time the resource usage and response time constraint satisfaction and to identify scheduling strategy for adapting the current scheduling of operators at run-time in order to decide how and by how much to replicate and place operators with respect to the evolution of the available Edge-Fog-Cloud resource capacities and DSPA application workload [30].

1.3 Contributions of the thesis

We are interested in operator replication and placement scheme that consumes as less as possible the Fog computational resources and Fog to Cloud network resources as well as satisfies any response time constraint set by a DSPA application. Figure 1.4 sketches the main contributions of the thesis, which are detailed in the sequel.

Edge/Fog/Cloud resource usage and application response time models. The first contribution of this thesis is a *resource usage model* for assessing the utilization of Edge-Fog-

Cloud resources allocated to the execution of a DSPA application. In this respect, a DSPA application is abstracted as a directed acyclic graph (DAG) of operators, while an Edge-Fog-Cloud architecture is abstracted as a hierarchical WAN of resources, where we identify for each resource (CPU, RAM, GPU of Edge/Fog/Cloud node or bandwidth of WAN link) its available and maximum capacities. Additionally, we introduce a *response time model* for a DSPA application, to which a response time constraint may be associated (see Step ① in Figure 1.4).

More precisely, we identify at a fine grained level the computational resource usage demand of each operator and the network resource usage demand for transmitting data streams between two operators executed at distributed nodes. Then, we propose a holistic resource usage cost model that can be used in two different cases. The first case takes into account that the DSPA application is statically deployed on dedicated Edge-Fog-Cloud resources (non shared resources). In this case, the request of using a resource is weighted by a static weight that captures only the maximum capacity of this resource. On the other hand, the second case takes in account that the DSPA application is dynamically deployed on non-dedicated Edge-Fog-Cloud resources (shareable resources). In this case, the request of using a resource is weighted by a dynamic weight that takes into account both the available and maximum capacities of this resource.

To estimate the response time of a DSPA application, we account for the transmission and propagation delays on each network link when sending and receiving data streams from the IoT devices at the Edge, considered as data sources, to the DSPA application sink deployed in the Cloud while traversing the Fog resource nodes on which the replicated operators of DSPA application can be placed. Furthermore, our response time model estimates the time required per each operator to process its input data stream in the specified window of an operator. Given the variability of IoT data stream rates and hence of the data stream size in the operator window, we abstract each operator with a queuing model to estimate the operator processing time (or operator latency).

Resource-aware scheduling of operators for IoT Edge analytics. The second contribution of the thesis is a set of new scheduling algorithms for statically deploying DSPA applications between the Fog and the Cloud with dedicated Edge-Fog-Cloud resources. In this respect, we use the version of resource usage model with static weights (Step ② in Figure 1.4). Then with this version of the resource usage model, we introduce a resource constraint satisfaction (RCS) algorithm that replicates and migrates operators from the Cloud to the Fog nodes. This baseline solution allows us to regulate the Fog computational resource usage and the Fog to Cloud network resource usage according to the evolution of IoT data stream rates, by minimally resorting to Fog resources in order to satisfy Cloud bandwidth constraints. RCS does not optimze the resource usage cost.

Next, we formulate the problem of replicating and migrating operators from the Cloud to the

Fog as a single objective optimization (SOO) problem. SOO essentially minimizes the combined cost of the Fog computational resource usage and the Fog to Cloud network resource usage while satisfying constraints on the usage of both types of resources. We observe that the SOO problem is a NP-hard problem. Towards its solution, we first formulate the SOO problem as an integer linear programming (ILP) model and exploit a mathematical optimization tool (CPLEX) to optimally solve it. However, our experimental results show that the SOO-CPLEX solution raises serious scalability concerns.

For this reason, we propose a heuristic algorithm, called SOO-H, which attempts to approximate the optimal SOO-CPLEX solution within a reasonable amount of time. Using thorough experiments, we demonstrate that SOO-H runs faster than SOO-CPLEX, regardless of the scale of the problem. Additionally, SOO-H reaches the optimal scheduling solution in most cases. Only in few cases SOO-H fails to find the optimal scheduling solution, but the approximation error is very small.

Resource-aware scheduling of operators for IoT Edge analytics with time constraints. The third contribution of the thesis extends the SOO problem with response time constraints set by DSPA applications (Step ③ in Figure 1.4). To this end, we exploit the response time model for estimating the DSPA application response time for a given placement solution between the Fog and Cloud nodes. Then, we formulate the related Time-based Single Objective Optimization (TSOO) problem and show that it can be mapped to a Job Shop Scheduling (JSS) problem which is NP-hard. In order to solve this TSOO problem, we exploit the meta-heuristic simulated annealing (SA) to formulate and solve the TSOO problem. We rely on SA as it has been proven in the literature to solve the instances of JSS problem. The drawback of TSOO-SA, is the high execution time of SA. Therefore, we propose a heuristic algorithm called TSOO-H to solve the TSOO problem in time efficient. TSOO-H approximates the optimal solution in a reasonable amount of time when comparing to TSOO-SA. However, it may fail to satisfy the response time constraint at highest data stream rates.

Adaptive scheduling of operators for IoT Edge analytics. The fourth contribution of the thesis extends the TSOO problem with the problem of dynamic deployment of DSPA application on the Edge-Fog-Cloud resources that can be shared among several other DSPA (non DSPA) applications. In this respect we use the version of the resource usage cost model with dynamic weights (Step ④ in Figure 1.4).

To this end, we consider a framework for threshold based monitoring approach of the workload of DSPA applications in order to trigger an adaptive rescheduling of the current operator placement solution whenever it is needed. To solve the extended TSOO problem instead of running from scratch the whole TSOO-H algorithm that may incur high execution cost in terms of both the execution time and operator rescheduling cost, we propose an adaptive version of

TSOO-H algorithm, called aTSOO-H.

aTSOO-H identifies the operators to replicate or to remove on the Fog in respect to the changes identified in the environment. By doing so aTSOO-H approximates the best feasible scheduling solution of operators at run-time with lower execution cost compared to TSOO-H, however with practically similarly resource usage cost and response time satisfaction ratio. In this respect, aTSOO-H is a best effort algorithm that provides feasible solution compared to TSOO-H algorithm.

Evaluation with recent related work shows that both TSOO and aTSOO-H provide significant solution in terms of successfully deploying DSPA applications across the Edge-Fog-Cloud resources shared among several application with lower overall resource usage cost.

1.4 Outline of the thesis

The remainder of the thesis is organized as follows.

In Chapter 2, we give the background of the DSPA application and the Edge-Fog-Cloud architecture. In Chapter 3, we analyze and classify the existing scheduling algorithms on resource allocation problem for DSPA application across distributed resource nodes (sensor network, peer resource nodes and Edge-Fog-Cloud architecture). To this end, we develop a general taxonomy that summarizes the main design choices of these existing solutions. Then we position our contribution with regard to the most relevant related works.

In Chapter 4, we introduce abstraction models of DSPA application and Edge-Fog-Cloud architecture. Based on these models we devise the resource usage cost model and DSPA application response time model for scheduling operators in the Edge-Fog-Cloud continuum.

Next, Chapter 5 presents the resource aware scheduling of continuous operators for IoT edge analytics that rely of the models presented in Chapter 4. Then follows Chapter 6 that presents the resource aware scheduling of continuous operators for IoT edge analytics with time constraints. The latter chapter is extended in Chapter 7 to account for the adaptive scheduling of continuous operators for IoT edge analytics. Then in Chapter 8, we conclude the thesis and present perspective for future work.

Chapter 2

Background

Contents

This chapter contains 20 pages.

2.1 Introduction

In this chapter, in Section 2.2 we present the core operators defined for data streams and describe the main steps in executing a DSPA application along with the main architectures of DSPA engines. In Section 2.3, we discuss the benefits of processing and analysing IoT data streams at the IoT network edge and introduce the challenges in pushing computations from the Cloud to the IoT network Edge.

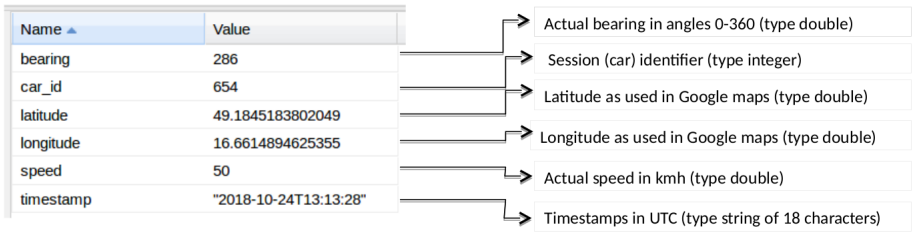


Figure 2.1: Example of a data tuple in a stream produced by connected vehicle [32]

2.2 Data Stream Processing and Analytics (DSPA)

Data Stream Processing and Analytics (DSPA) relies on the principle of online computation to mine data stream in near real time [22]. 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 a operator are constantly updated each time new data tuple of input data streams are processed [7].

2.2.1 Data Streams

A data stream is an unbounded sequence of tuples [31]. In general, all tuples of a data stream share the same schema. As shown in Figure 2.1, a tuple is defined as list of attribute-value pairs where each attribute has a name and a type (e.g., integer, double, string, etc.). We denote the schema of a generic data stream as $(A_1, A_2, \ldots, A_n)$ and we refer to attribute A_i of a data tuple d by $d.A_i$.

A data tuple in a stream is timestamped. Depending on the DSPA engine [33], timestamps can be set in different ways: (i) the *event time* is related to time that the data tuple was generated, (ii) the *ingestion time* is related to the time that the data tuple entered into the system (DSPA engine, DSPA application) and (iii) the *processing time* is related to the time that the data tuple was processed by an operator. The event time and the processing time assume that the data stream sources and the nodes hosting the DSPA application use a clock synchronization protocol like NTP [34]. The ingestion time can be used in the case clock synchronization protocol is not possible.

2.2.2 Time-based and Tuple-based Windows

Because of the infinite nature of the data stream, the mechanism called *window* has been introduced for setting an operator with a flexible bounds on the unbounded data stream in order to fetch a finite, yet ever changing set of data tuples, which may be regarded as a temporary relation [35]. Formally, it is defined as follows:

$$W\{\forall t \geq t_0 | Wstate \leftarrow \{d \in Sin | p(d,t) = \text{ True }\}\}(Sin, Wstate, Wtype, Wsize, [Wadvance], t, t_0) \tag{2.1}$$

where $Wtype$ is the type of a window that can be time-based or tuple-based. The former is defined over a period of time (e.g., data tuples received in the last 5 minutes) while the latter is defined over a fixed number of data tuples (e.g., last 100 data tuples). For both type, $Wstate$ is the state of the window defined as the number of data tuples that it contains at every time instant t.

$Wsize$ is the *size* of the window that corresponds to the window boundary, e.g., 5 minutes for the time-based window and 100 data tuples for the tuple-based window. The parameter $Wadvance$ called *slide* is optional and used to process the data tuples in a window of size $Wsize$ by slide of $Wadvance$. Sin is the input data stream from which the input data tuples arrive. t_0 is the time at which the windowed operator has been deployed while t is the current time. Finally, p is a condition to verify if a data tuple d satisfies the window size (and the slide size) boundary before to be included in the window state $Wstate$.

Two kinds of windows are widely used in the recent generation of DSPA engines, namely, *time-based* and *tuple-based* windows distinguished according to the unit in which window state is determined [35]. It worth noting that in our work we do not cover the *session* window. Unlike the formal definition of windows introduced bellow, a session window does not rely on a static boundary. It actually depends on a defined session period which can be static or dynamic (see [36] for further details).

2.2.2.1 Time-based windows

In time-based windows the timestamps of data tuples are checked for inclusion within a specified temporal boundary. This requirement is expressed by means of a scope function that may be defined for each window type. In essence, at every time instant the scope function returns the window time boundary and not the actual window state. This scope function take as parameter the window size and the window slide. We present the *Time-based sliding* window and *Tumbling* window as the most representative time-based window types. For further reading on the time-based windows, we refer reader to [37].

Time-based sliding windows consider the invariable temporal extend of the window called window size $Wsize$, the progression temporal step, called sliding size $Wadvance$. In this respect, the window contains the set of data that arrive within the last $Wsize$ time units, so that the set of data within the window are processed every $Wadvance$ time units.

Let consider t_0 as the time at which the operator is deployed considering a time-based sliding window W, we assume the time window size has no delay with regard to the current time t. Thus, the scope function of this window can be defined as a function of time:

$$Wscope(t) \leftarrow \begin{cases} [t_0, t] & \text{if } t_0 \leq t < t_0 + Wsize \wedge mod((t - t_0), Wadvance) = 0 \\ [t - Wsize + 1, t] & \text{if } t \geq t_0 + Wsize \wedge mod((t - t_0), Wadvance) = 0 \\ Wscope(t-1) & \text{if } mod((t - t_0), Wadvance) \neq 0 \end{cases} \tag{2.2}$$

where t_0 and t are timestamps, $Wsize$ and $Wadvance$ are sizes of time interval and sliding where $Wsize > 0$ and $Wadvance > 0$.

For every time instant t, the window state contains the data tuple of the data stream Sin whose the timestamp is in the time interval of the scope function:

$$W\{Wstate \leftarrow \{d \in Sin | d.ts \in Wscope(t)\}\}(Sin, Wstate, Wtype, Wsize, Wadvance, t, t_0) \tag{2.3}$$

where ts is the timestamp of the input data tuple d.

For a time sliding window where $Wsize > Wadvance$, an overlapping of data tuples is observed between two successive states of the time sliding window. Thus, a subset of their data tuples remains the same across states.

Time tumbling windows consider only the invariable temporal extend of the window called window size $Wsize$ so that the window state contains the set of data tuples that arrive within the last $Wsize$ time units, that are processed every $Wsize$ time units. In this respect, there is no data tuple overlapping, each data tuple belongs to only one window state. A tumbling window can be seen as a special case of the time-based sliding window where $Wsize = Wadvance$. Thus, the time tumbling window is formally defined as following:

$$W\{Wstate \leftarrow \{d \in Sin | d.ts \in Wscope(t)\}\}(Sin, Wstate, Wtype, Wsize, Wsize, t, t_0) \tag{2.4}$$

2.2.2.2 Tuple-based windows

The window state is determined by counting the most recent data tuples. Thus, $Wsize \in \mathbb{N}$ is a natural number (not determined in terms of time unit). We present the tuple-based window and Partitioned window. For further reading on the tuple-based windows, readers are referred to [37].

Tuple-based window At every time instant t, a tuple-based window covers the most recent $Wsize$ data tuples d of the input data stream Sin:

$$\begin{aligned} &W\{Wstate \leftarrow \{d \in Sin : (\exists t_1 \leq t \wedge \|d \in Sin : t_1 \leq d.ts \leq t| \leq Wsize) \\ &\wedge (\forall t_2 \leq t_1 \wedge |d \in Sin : t_2 \leq d.ts \leq t| > Wsize)\}\}(Sin, Wstate, Wtype, Wsize, t) \end{aligned} \tag{2.5}$$

where t_1 and t_2 are timestamp values. The former bounds the most recent data tuples while the latter bounds the data tuples that can be considered as outdated. In this respect, in order to group the set of $Wsize$ last data tuple, the tuple-based window starts from the current time t, selects data tuple d by going steadily backwards in time until the $Wsize$ data tuples are collected.

Partitioned window (operator) splits its input data stream Sin into a set of data substreams $L = \{A_1, A_2, \cdots, A_k\}$ according to a grouping over data tuple attributes and each substream corresponds to a combination of values $D_k = (a_1, a_2, ..., a_k)$ on the grouping attributes. Then, the tuple-based window is applied on each substream and the union of the resulting data tuple constitutes the window state $Wstate$. Formally, it is defined as following:

$$
\begin{aligned}
&W\{Wstate \leftarrow \bigcup_{A_k \in L} \{d \in Sin : \forall A_k = a_k \wedge a_k \in D_k \\
&\wedge (\exists t_1 \leq t \wedge \|d \in Sin : d.A_k = a_k \wedge t_1 \leq d.ts \leq t| \leq Wsize) \\
&\wedge (\forall t_2 \leq t_1 \wedge |d \in Sin : d.A_k = a_k \wedge t_2 \leq d.ts \leq t| > Wsize)\}\} \\
&(Sin, Wstate, Wtype, Wsize, L, t)
\end{aligned}
\tag{2.6}
$$

It worth noting that, the timestamp attribute is not involved in grouping. Thus, tuple-based windows may be considered as a special case of partitioned windows where all data tuples of the data stream get assigned to a single partition [37].

2.2.2.3 Out-of-order data tuples in data streams

Both time-based and tuple-based windows rely on a time notion to specify the contents of a current window. However, how we can ensure that a set of incoming data tuples corresponds to a specific window when they arrive out-of-order? This is frequently the case of data tuples from IoT streams as devices they produce them may stay off-line due to a network issues and thus send out-of-order data tuples after some time. This issue is usually handled by DSPA engines using a watermarking heuristic. The wisdom behind the watermark is to balance between including as much late data as possible and not delaying window processing too much [33].

2.2.3 Operators

Typical operators specifying a DSPA application are analogous to relational algebra operators and can be classified as *stateless* or *stateful* [35, 38]. Stateless operators (e.g., Map, Union and Filter) do not keep state across data tuples and perform their computation solely based on each input data tuple. Stateful operators (e.g., Aggregate, Join and Cartesian Product) perform operations on sequences of data tuples.

The result of a operator is a data tuple whose constituents may differ from the input data tuple both in terms of content and structure [31]. An operator is characterized by a *selectivity* defined as the the ratio between the number of data tuple (or the number of data tuple attributes) it produces and the number of data tuple (or the number of data tuple attributes) it consumes [39].

In the following, we present formally the typical operators including Map, Filter, Aggregation, Union and Join [40].

2.2.3.1 Filter

The filter operator is used to select input data tuples according to a set of predicates and route them over one or more output data streams.

In practice, filter operation involves selection or projection. In this respect, the selection can be defined as the operation of selecting data tuple d from input data stream Sin that satisfies a given predicate p on its attribute. Thus, the selection operator is formally defined as following:

$$\sigma\{Sout \leftarrow \bigcup\{d \in Sin : p(d) = TRUE\}\}(Sin, p) \tag{2.7}$$

On the other hand, the projection can be defined as the operation of selecting attribute of data tuple d from input data stream Sin in order to produce output data stream $Sout$ constituted with data tuple with new attribute schema.

$$\pi\{Sout \leftarrow \bigcup\{d_{out} \leftarrow \bigcup_{A_k \in d} \{d \in Sin : d.A_k = P.A_k\}\}\}(Sin, P(A_1, ..., A_m)) \tag{2.8}$$

P contains the attributes that will constitute the schema of the output data tuple d_{out} in the output data stream $Sout$. In this setting the number of attributes of the output data tuple d_{out} should be less than the number of attributes of the input data tuple d.

2.2.3.2 Map

The Map operator is used to transform each input data tuple to another output data tuple via set of user defined functions. The schema of the output data tuple may be different from the schema of the input data tuple but the timestamp of the former is preserved in the latter. Formally the Map operator is defined as follows:

$$\Xi\{A'_1 \leftarrow f_1(din), A'_2 \leftarrow f_2(din), \cdots, A'_n \leftarrow f_n(din)\}(din, dout) \tag{2.9}$$

where din is the input data tuple defined by the schema $(A_1, \cdots A_m)$, $dout$ is the output data tuple defined by the schema $(A^{'}_1, \cdots A^{'}_n)$ and $m \neq n$. Finally ,$f_1, \cdots f_n$ are the user defined functions.

2.2.3.3 Union

The union operator merges multiples input data streams sharing the same schema into a single output data stream by using the first in first out (FIFO) policy. It is formally defined as follows:

$$\bigcup\{Sout \leftarrow \{d \in (\cup_{i=1}^{n} Sin_i)\}\}(Sin_1, \cdots, Sin_n, Sout) \tag{2.10}$$

where $Sin_1, \cdots, Sin_n$ are the input data streams, $Sout$ is the output data stream.

2.2.3.4 Aggregation

The aggregation operator is used to compute aggregate functions such as count, sum, average, etc. over a set of data tuples in a window. More formally, it is defined as follows:

$$Ag.\{W, A'_1 \leftarrow f_1(W), \cdots, A'_n \leftarrow f_n(W), [Group - by = (A_1, \cdots, A_m)]\}(Sin, Sout) \tag{2.11}$$

where W is the window used by the operator that can be time-based or tuple-based. Sin is the input data stream whose data tuples fed to the various windows types. When a window reaches its time- or tuple-based boundary, the aggregate functions $f_1(W), \cdots f_n(W)$, are triggered on its input data tuples and produce a single output data tuple over the output data stream $Sout$. The schema of the output data tuple is different from the one of the input data tuples and the timestamp of the output data tuple is the one associated to the earliest data tuple in the window.

A Group-by operation is optional and used to partition the data tuples of the input data stream Sin. Assume for example $Group - by = A_i$, where A_i is an attribute of the input data tuple. Then, the operator handles separate windows for each possible value of A_i.

2.2.3.5 Join

The Join operator is used to match data tuples from multiple input data streams. It is formally defined as follows:

$$\bowtie \{W, p\}(Sin_r, Sin_l, Sout) \tag{2.12}$$

where Sin_r and Sin_l are two input data streams refereed as right and left respectively while $Sout$ is the output data stream. p is a predicate for matching a pair of data tuples, one from each input data stream.

The join operator keeps separate windows Wr, Wl for each input data stream. Data tuples arriving on the left (respectively right) input stream are used to update the right (respectively left) window. If the window is a *logical* one, upon arrival of data tuple $din \in Sin_l$, the window Wr is updated by removing all data tuples d such that $din.ts$ - $d.ts$ is higher or equal to the

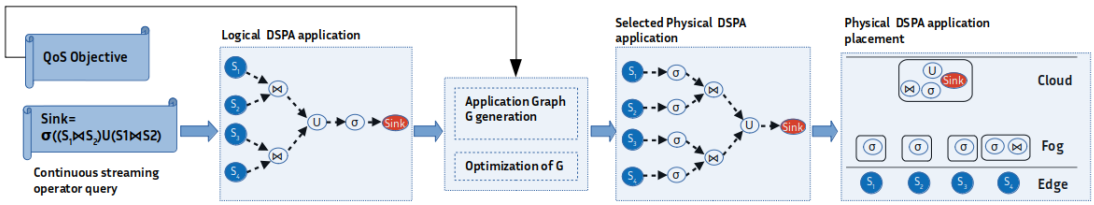


Figure 2.2: From deployment to execution of DSPA application

window size [37]. If the window is a *physical* one, upon arrival of data tuple $din \in Sin_l$, the window Wr, if it is full, it is updated by removing the earliest data tuple. After window update, predicate evaluation and output propagation for input data tuples over the right stream are performed in a similar fashion.

It worth noting that, the Cartesian product operator (X) is defined and works like the join operator with a predicate always true.

2.2.4 DSPA applications in Action

To implement a DSPA application, developers use an SQL-like continuous query [41, 42] which is submitted to a *DSPA engine* capable of executing the operators on an underlying computing infrastructure (on the Cloud, a data center, etc.).

In this respect, DSPA engine parses a continuous SQL query to generate an associated logical application graph. It is essentially a *directed, acyclic graph (DAG)*, where the vertices capture operators, and the edges capture the data streams flowing between them [7]. This logical application graph is then optimized in order to fulfill certain *quality of service (QoS)* associated with the DSPA application. In particular, graph rewritings is applied w.r.t. an optimal usage of the network and computational resources of the nodes chosen to execute the operators. During this *optimization*, several candidate physical placements of operators are generated, and the one that fulfill optimally the QoS objective is selected to be deployed on the resource nodes. Figure 2.2 shows the main steps in executing a DSPA application.

2.2.4.1 Quality of Service

A DSPA engine should respect the QoS requirements of a DSPA application to support real time data stream processing. This QoS is typically expressed in terms of high throughput [43] and low end-to-end latency (response time) [8]. To foster DSPA application availability, a DSPA engine could also optimize the resource usage cost [22] (e.g., for monetary purpose) in terms of CPU/RAM usage, network bandwidth usage and energy consumption.

Table 2.1: Optimization Policies on DSPA application

Policy	**Change topology**	**Adapted application**	**Occurring time**
Operator reordering	Yes	Logical	depends
Replication	Yes	Logical	depends
Placement	No	Physical	depends

2.2.4.2 Optimization policies

[44] surveys several optimization policies used to reconfigure a DSPA application graph in order to satisfy a QoS objective. In our work, we consider two main optimization policies: (i) *graph rewriting* that includes the operator reordering and the operator replication and (ii) *operator placement*. We describe bellow these policies by highlighting in Table 2.1 whether they change the topology of the DSPA application and whether they are applied to adapt its previously scheduled deployment, consequently whether the optimization policy can occur respectively at deployment time or at run-time.

Graph rewriting enables to rewrite a DSPA application graph into an equivalent one that when executed on a number of allocated computational resources can satisfy a QoS objective. It essentially rewrites operators using well-known equivalence rules of relational algebra [45]. We present in the following few of these rules where we consider S as the input data stream and p as the predicate associated with an operator:

1. Conjunctive selection can be rewritten as a sequence of selections:

$$\sigma_{p_1 \wedge p_2}(S) \equiv \sigma_{p_1}(\sigma_{p_2}(S)) \tag{2.13}$$

2. Selection over Join can be rewritten as Join:

$$\sigma_p(S_1 \bowtie S_2) \equiv S_1 \bowtie_P S_2 \tag{2.14}$$

3. Conjunctive selection can be distributed over Join:

$$\sigma_{p_1 \wedge p_2}(S_1 \bowtie S_2) \equiv (\sigma_{p_1}(S_1)) \bowtie (\sigma_{p_2}(S_2)) \tag{2.15}$$

4. Filter, Map, Aggregation, etc. can be distributed over Union:

$$\sigma_p(S_1 \bigcup S_2) \equiv \sigma_p(S_1) \bigcup \sigma_p(S_2) \tag{2.16}$$

The translation of a logical DSPA DAG to a physical one that will be executed on the resource nodes relies on two main rewriting policies described as follows [8]:

Definition 1. (Reordering) "moves an operator from one node to another by following the topological order of operators in the DAG. For instance, a selection can be moved near to the source of the DAG to reduce the amount of data tuples of two streams that are joined afterwords (see Formula 2.14). As the performance gains achieved by this policy depends on the actual selectivity of operators which may change at run-time, operators reordering at deployment time may not always be sufficient".

Definition 2. (Replication) "replicates a operator in the DAG, so that each replica can process a portion of the operator input data stream to enhance parallel data processing [7]. As the performance gains achieved by this policy heavily depends on the actual dynamics of the DSPA application workload, operators replication should be decided at run-time".

Changing at run-time a running DSPA DAG composed of stateless operators is easy. We need to simply replace the old by the new execution. This is not the case of graphs composed of statefull operators. In this case, the new execution of the DSPA DAG should preserve the execution state of the old one. For example, the window state of the join operator in the application graph $\sigma(S_1 \bowtie S_2)$ can not be recovered by the application graph rewrites as $\sigma(S_1 \bowtie S_3)$. Nevertheless, in this thesis we do not explicitly cover the reconfiguration of DSPA application constituted by state-full operators to manage state migration.

operator placement In order to deploy a physical DSPA DAG on our computing infrastructure, we need to select the nodes on which operators will be executed to respect the QoS objective of the DSPA application and eventually an overall optimal resource usage. Placement decisions are usually made once at deployment time [22–24]. Some placement algorithms [8, 46, 47] continue to be active also at run-time, in order to response to changes in the DSPA application workload or changes in the availability of the allocated resources.

2.2.5 DSPA Engines Architecture

We consider general DSPA engines that leverage the relational model of Data Base Management Systems (DBMS) for specifying DSPA applications using SQL-Like continuous queries. Unlike one-shot operators of traditional DBMS, once a operator is deployed, its results are computed each time a new data tuple becomes available in its input [7]. DSPA engines are typically categorized under 3 generations [48] while a fourth one [43] has been recently proposed to accommodate the growth of IoT applications.

First generation of DSPA engines heavily relies on DBMS technology extended to process long running queries in a centralized setting. They have been built to run as standalone prototype or as an extension to an existing DBMS [48]. The notorious examples of DSPA engines of this generation are Aurora [38] and TelegraphCQ [49].

Second generation of DSPA engines introduces distributed processing of operators to take the advantage of abundant computational resources available in clusters. In this respect, these DSPA engines have been extended to support more expressive operators [50] but also additional services such as fault tolerance [51] and adaptive query processing [52]. Borealis [51], CEDR [50], CAPE [52], etc. are typical examples of this generation.

Third generation of DSPA engines is influenced by the trends of massively parallel and distributed systems for cloud computing. Beside SQL-like queries, engines of this generation are able to support a wide range of complex jobs like iterative machine learning (ML), interactive queries and online processing on different data modalities (record or graph data). Well known DSPA engines are among others Apache Storm [53], Apache Flink [54], Apache Kafka [55], Apache Spark [56], etc. It is worth noting that such DSPA engines can also be deployed on heterogeneous computing systems such as Edge/Fog nodes to process data streams near to their source of creation.

Fourth generation of DSPA engines aims to overcome the limitations of Edge/Fog computing architectures in terms of constrained resources that can impair the performance of DSPA applications [57]. In particular, this generation of DSPA engine becomes aware of the Edge/Fog resources at the IoT network edge that could execute an application eventually in cooperation with a cloud-based DSPA engine. Nevertheless, the development of such DSPA engines are still in their infancy. Apache Minifi [58] is a framework for deploying data collection applications on resource-poor nodes characterized by a lightweight energy footprint. It aims to supplement the core tenets of NiFi [59] as a powerful and reliable DSPA engine designed for resource-rich computing nodes. IBM Apache Edgent [60] is a stream processing programming model and lightweight run-time framework to support IoT data analytics at the edge nodes or the gateway. Edgewise [61] and Resense [62] are recent efforts to support IoT data analytics on resource-constraint devices at the edge or fog nodes. Furthermore, several commercial DSPA engines such as Amazon IoT greenglass [63] aim to extend the cloud IoT analytics to local IoT devices, enabling devices to efficiently collect and analyse data closer to data sources. Finally, FogHorn [64] aims to enable Artificial Intelligence (AI) at the IoT edge for high volumes, varieties and velocities of live sensors and machine data, that is optimized for resource-constrained devices. More recently, NebulaStream [65] is emerging as the major engine to overcome the limitation of the Edge/Fog computing. It is designed to incorporate all computing resources, even outside the cloud (i.e. IoT network edge), and apply processing wherever possible.

2.3 Edge/Fog Cloud continuum

In this section, we describe a multi-tier architecture that introduce between the Cloud and the IoT devices, Edge and Fog layers.

2.3.1 Definitions and motivation

According to National Institute of Standard and Technology (NIST) [66]:

> **Definition 3.** (The Cloud computing) "is defined as model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction".

The main Cloud providers are among other OVH [67], Google Cloud [68], Amazon Web Service (AWS) [69], etc. They mostly offer Cloud services with a pay-as-you-go model [57, 70]. The Cloud is the prevalent environment for executing DSPA applications that requires to transfer the data streams produced by IoT devices over the WAN links. This Cloud-based IoT Analytics architecture (see Figure 1.1a) was enabled by the increasing connectivity of IoT devices (i.e., things) and the practically unlimited Cloud resources offered to host the third generation of DSPA engines.

In particular, the Cloud provides [66]: (i) On demand self-service to enable Cloud resources to be provisioned on-demand automatically without human interaction; (ii) Broad network access to enable IoT devices to access via the internet to the Cloud resources in a seamless manner; (iii) Resource pooling to serve seamlessly multiple Cloud consumers; (iv) Rapid elasticity to scale according to the consumer demands by allocating or releasing Cloud resources automatically; and (v) Measured service to enable transparency to Cloud provider and consumers by completely monitoring Cloud resource usage. As result, the Cloud ensures high availability of DSPA applications and application owners are charged only on the basis of the Cloud resources actually consumed by the application.

The terms Edge and Fog computing are often used interchangeably [71]. However, in this thesis we consider them as different layers where the boundary between the two is tiny [72]. In particular, we consider that Edge/Fog layers provide complementary computational and network resources to the Cloud enabling IoT edge analytics in the Edge-Fog-Cloud continuum as depicted in Figure 1.1b.

In this context, we refer to the definitions of the Edge and Fog as provided by the National Institute of Standards and Technology [73]:

Definition 4. (The Fog computing) "is a horizontal, physical or virtual resource paradigm that resides between smart end-devices and the traditional Cloud. This paradigm supports vertically-isolated, latency-sensitive applications by providing ubiquitous, scalable, layered, federated, and distributed computing, storage, and network connectivity".

Definition 5. (The Edge computing) "is referred as the IoT network encompassing the smart end-devices and their users, to provide, for example, local computing capability on a sensor, metering or some other devices that are network-accessible".

Fog computing is generally a geographical distributed (or not virtualized) platform providing computational, network and storage service between the IoT devices at the Edge and the Cloud. While the Edge computing is highly distributed including all the IoT devices augmented with computational and networking capabilities and connected to the Cloud through a Fog computing nodes.

2.3.2 Benefits

Event though that the initial concepts of Edge/Fog computing aiming to process data at the IoT network edge were formulated more than a decade ago, its main motivations are still prevailing today. In the context of the recent growth of IoT industry, it evenly shows high benefits in terms of (i) preventing network congestion; (ii) reducing the network delay, and (iii) ensuring privacy of sensitive data. It worth noting that ongoing research works intend to demonstrate that Edge/Fog computing may support sustainability in terms of electricity consumption and carbon footprint. However, further insights from large-scale measurement exercises are required to make a more informed case [13].

2.3.2.1 Reduce network delay

Reducing the network delay by processing IoT data streams at the IoT network edge is one of the motivations in favor of the Edge/Fog computing. However, different technology providers may consider the network delay in different ways. Therefore, for the sake of clarity we define in the following what should constitutes the network delay.

The network delay includes the propagation delay on the network link medium that depends on the distance (number of hops) between the source node and the destination node. The propagation delay is the time it takes to transmit bits of data between a source node and a destination node and it is independent on the data size [74,75]. However, it depends on the type of the network link medium, the distance between the connected endpoints and is limited by the speed of the light. For instance if a source node and a destination node are in the same building at the distance of 200m, the propagation delay will be 1 microsecond. However, if they are located in different countries at a distance of 20000 Km, the delay is in order of 0.1 second [76].

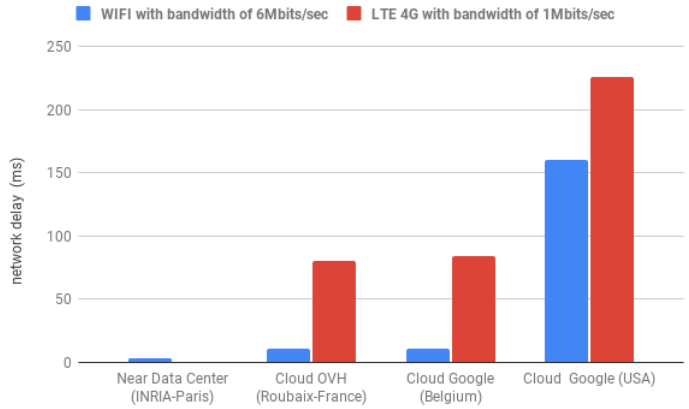


Figure 2.3: Network delay differences from the IoT network edge to the distant Cloud

Additionally, the network delay includes the transmission delay that depends on the size of the data to transmit and the available network bandwidth between the source and destination nodes. The available network bandwidth depends on several factors including number of active sessions, transmission capacity of the link (nominal network bandwidth capacity), medium access control (MAC) access delay, etc. [76].

Finally, the network delay includes also the processing and queuing delays of a packet at the intermediate routers [37, 77, 78]. The latter can be big in case of congestion at the router but in general these delays can be considered as constant [13].

If we assume that the processing and queuing delays of a packet at the intermediate routers are constant, the main components that constitute the network delay become the propagation and transmission delays.

Figure 2.3 shows the benefits in terms of network delay when sending a ping message with packet size of 1400 bytes from a Raspberry Pi 3 located at INRIA Paris to a closest VM server located in the INRIA Paris data center with average 3ms of network on WIFI , versus sending to a VM server located respectively on OVH site in Roubaix in France with average 10ms on WIFI and 79ms on 4G, and Google Cloud site in United State of America (USA) with average 150ms of network delay on WIFI and 255ms on 4G. These results show that processing data stream at the network edge will support time sensitive of DSPA application with a low network delay between 3ms to 10ms.

Although the theoretical 1ms of network delay of 5G, real deployment in USA and United Kingdom have demonstrated respectively 30ms and 20ms of network delays to the distant Cloud. According to [13], such network delay can support many time sensitive applications. However, they will not be adequate to support applications requiring constrained response time (sub millisecond), such as those involving autonomous cars or robots.

In this respect, by combining communication technology providing low network delay like 5G and the Edge/Fog computing is one of the solutions to further reduce the network delay for supporting both time sensitive and highly time sensitive applications.

2.3.2.2 Prevent network congestion

Sending data stream from the IoT devices to the distant Cloud require traversing WAN links with several hops. According to Varghese et al [13] the network bandwidth between two VM located in the same AWS data centers have been demonstrated to be on average 900Mbps. However when the WAN is involved, the network bandwidth to the same VMs was between 30Mbp-160Mbps.

For a DSPA application relying on intensive IoT data streams, sending huge volume of data stream to the distant Cloud may overwhelm the network resources to reach the Cloud and cause network congestion that can consequently increase the network delay. Additionally, most Cloud providers limit the bandwidth when the total data transfer reaches a certain threshold [13].

In this respect, relying on the Edge/Fog computing enables to partially process data streams and sending to the Cloud reduced size data stream, thus preventing network congestion.

2.3.2.3 Ensure data privacy

With the high growth of the IoT industry, the IoT devices collect and produce data that are useful for consumers, businesses and public sector policy-making. In this context privacy arise naturally as IoT data has to be transmitted to the distant Cloud while non anonymous data can sometime provide real insights into an individual behavior, health or relationships.

For example in Figure 2.1 data tuples related to driving speed and location can be used against car owners by the car insurance companies. The latter, can set high insurance price according to whether car has a sportive driving behavior. In this respect, by leveraging Edge/Fog computing as a privacy preservation means [79], IoT data can be aggregated closer to where they are created and send to the Cloud only the average values per district or street. Nevertheless, the Edge/Fog providers should be trustworthy.

2.3.3 Challenges

Even though that IoT Edge Analytics enables to reduce the network delays, prevent network congestion and ensure data privacy, the environment on which a DSPA application is deployed influences drastically the associated QoS in terms of security, response time (i.e., the sum of network delays and processing time), throughput, and resource usage cost (i.e., CPU, memory, storage, energy, bandwidth, etc.). In the following, we revisit the challenges that come with IoT edge analytics.

2.3.3.1 Resource heterogeneity at Edge/Fog layers

In IoT cloud analytics, application providers can scale cloud resources (CPU, memory, storage, bandwidth, etc) on demand by using a pay-as-you-go model [19]. However in IoT Edge Analytics, the Edge/Fog computing consider different execution models [31] and rely on geographical distributed heterogeneous nodes such as mobile or fixed IoT devices (e.g., camera, connected vehicle, smartphone, etc.), small data centers, routers, wireless base stations, etc. that come with stringent resource constraints in terms of limited computational resources (e.g., CPU, RAM, etc.) and limited power supply (e.g., rechargeable batteries, solar energy, etc.) that may have to be shared amongst several DSPA applications. Therefore, computing resource allocation becomes a prominent challenge. Insufficient computational resource allocation leads to workload imbalance, increase of computation time, QoS violations, and affects energy consumption of Edge/Fog nodes [13, 80] In this respect, an efficient resource allocation mechanism is required [81].

Even though significant developments are achieved, techniques are very diverse, to name [19, 81] propose taxonomies for resource management solutions in Edge-Fog-Cloud. Nevertheless, the problem is still very challenging which always presents a hot issue for the research community.

2.3.3.2 Dynamic Workload

The workload of a system is defined as the set of all inputs received by the system from its environment during a time period. Long running DSPA application is characterized by a dynamic workload owing to the spatial and temporal dynamics of IoT device distributions. Thus, the IoT data stream rate can be dynamic until it influences the workload behavior. Clearly a dynamic workload has an impact on QoS requirement of DSPA application: higher workload may incur high usage of computational resources, energy and bandwidth along with high network and processing delays. Therefore, dynamic scheduling of DSPA operators according to the evolution of the workload is necessary to ensure the associated QoS requirements. However, it requires appropriate model and mechanisms to decide, how to schedule, when to schedule and by how much [30, 81]

2.3.3.3 Privacy and Security

Today's IoT devices rise many vulnerabilities due to the lack of adoption of well-known security techniques, such as encryption, authentication, access control, and role-based access control. The reason for this lack is that existing security techniques, tools, and products may not be easily deployed to IoT devices because of, the variety of hardware platforms and limited computing resources of devices. Although the concept of processing IoT data at the network edge by providing computing resource close to IoT devices provides better structure to enforce data privacy [79], the distributed architecture of IoT edge analytics increases in fact the dimension

of attack vectors [82]. Therefore, revisiting and extending existing privacy and security techniques to address the specificities of IoT analytics systems entails not only many scientific and engineering challenges but also public policy challenges.

2.3.4 Framework overview to support IoT Edge Analytics

Edge/Fog computing paradigms are still struggling to find an official implementation. However, several efforts continue to be made in order to provide simulated or real prototype implementations. Thus, we survey the main prototype implementations that we present a brief summary in Table 2.2, where we consider: (i) architecture to describe the number of intermediate layers between the IoT devices and the Cloud; (ii) App. type to describe the type of the application whether it is based on DSPA, micro-service (MS); interconnected application modules (AppM), etc. that the prototype supports; (iii) mobility to describe whether the prototype includes mobile resource by enabling dynamic discovery and federation of mobile resources to the global prototype architecture; (iv) monitoring to describes if the monitoring is supported by the prototype to give insight of resource state at run-time; and (v) Type: to define whether the prototype is a simulator or is implemented on real resources as a private or open source platform.

In the sequel, we briefly present the main efforts to implement or simulate an Egde-Fog-Cloud computing platform [83].

2.3.4.1 KubeEdge

KubeEdge [84] is an open source system for extending native containerized application orchestration at the IoT network edge. It is built on top of Kubernetes and provides fundamental infrastructure support for network, application deployment and metadata synchronization between the Cloud and the Edge. Kubernetes is an open-source system that enable to automatically deploy and manage large scale cloud applications using containers such as docker. The communication of module in KubeEdge is based on the decouple communication protocol called MQTT protocol. In this respect a mobile node can connect and disconnect without significant impact of the overall KubeEdge.

2.3.4.2 iFogSim

iFogSim [85] is a toolkit based on the well adopted CloudSim framework. CloudSim [86] is the most used framework for modeling and simulating the Cloud environment. Thus, iFogSim extends the basic abstract classes of Cloud to offer a framework for modeling and simulating the Edge/Fog computing environment with large number of layers between the IoT devices and the Cloud. iFogSim applies Sense-Process-Actuate and distributed data-flow model while simulating any application scenario in Fog computing environment. It facilitates evaluation of end to end latency, network congestion, power usage and computational resource usage. However

iFogSim does not yet support run-time reconfiguration of simulated application. In this respect MobFogSim [87] extends iFogSim to support reconfiguration of simulated application.

2.3.4.3 Enorm

Enorm [88] is a framework providing a 3-tier architecture composed of Cloud, Edge and IoT devices. The management is centralized in the Cloud. IoT devices offload tasks to Edge nodes when necessary, either due to user mobility or QoS (e.g. violation of latency constraint). Enorm dynamically allocates resources by an auto-scaling mechanism which scales up/down the resources, considering the network latency and task execution time. However Enorm does not support dynamic reconfiguration of application.

2.3.4.4 FogFlow

FogFlow [89] is programmable Fog computing environment on resource network divided vertically in 3 layers including IoT devices layer, Fog/Edge layer and Cloud layer. To support openness and interoperability of IoT applications, FogFlow uses the data flow model for modeling DSPA application in which operators are define as dockerized application based on the standard NGSI. NGSI [90] is an open standard that enables to define both data model and communication interface to exchange contextual information between applications (operators) via context broker. In this respect, FogFlow uses 3 logical views to operate on a geo-distributed resource network that include service management, data processing and context management. (i) Service management is deployed in the Cloud and includes task designer (TD), docker image repository (DIR) and topology manager (TM). TD provides the web-based interface to enable a developer to design, submit, monitor and manage operators of a DSPA application. DIR manages the images of all dockerized operators submitted by the developer. TM is responsible for orchestration and mapping operators as tasks to workers (Cloud and Edge/Fog nodes). The service orchestration has the objective to minimize the network bandwidth usage without overloading the workers. (ii) The data processing view manages to worker to perform data processing task assigned by TM. (iii) Context management view enables to establish data flow across the tasks via NGSI [90].

2.3.4.5 OpenStack

OpenStack [91] is an open-source platform designed for deploying a Cloud computing by using virtualization through infrastructure as a service model. Even though that the main goal of OpenStack is to support the cloud computing in data center. Today's OpenStack is extended to support an Edge-Cloud computing use case by providing a flexible and modular design [92].

2.3.4.6 EdgeNet

EdgeNet [93] is a public Kubernetes cluster dedicated to network and distributed systems research, supporting experiments that are deployed concurrently by independent groups. Its nodes

Table 2.2: Summary of Edge/Fog platforms

	Architecture	**App. type**	**Mobility**	**Monitoring**	**Type**
KubeEdge [84]	2 layers	MS, DSPA, AppM	Supported	Supported	Open source platform
iFogSim [85]	extensible layers	MS, DSPA, AppM	Not supported	Supported	Simulation platform
MobFogSim [87]	extensible layers	MS, DSPA, AppM	Supported	Supported	Simulation platform
Enorm [88]	3-layers	MS, DSPA	Supported	Supported	Open source platform
FogFlow [89]	3-layers	DSPA	Supported	Supported	Open source platform
OpenStack [91, 92]	2-layers	DSPA, MS, AppM	Supported	Supported	Open source platform
EdgeNet [93]	extensible layers	DSPA, MS, AppM	Not supported	-	Open source platform
Pan et al. [94]	3-layers	Smart grid	Supported	Supported	Research prototype

are hosted by multiple institutions around the world. It represents a departure from the classic Kubernetes model, where the nodes that are available to a single tenant reside in a small number of well-interconnected data centers. The free open-source EdgeNet code extends Kubernetes to the edge, making three key contributions: multi-tenancy, geographical deployments, and single-command node installation.

2.3.4.7 Research prototype

Due to the lack of well defined architecture that include both Edge, Fog and Cloud computing, researchers propose prototype implementation of such architecture that focus on specific application use cases. For instance, (and not limited to) Pan et al. [94] propose a prototype architecture of Fog computing for smart grid based application. In essence, it provides an architecture that strengths the coordination between Fog nodes to reduce smart grid based application latency. This prototype takes into account mobile devices as well as resource usage monitoring. However it is designed for specific application and does not support DSPA application.

2.4 Conclusion

In this Chapter, we described the main functionality of DSPA applications for processing data streams on the fly via a series of operators. More precisely, a DSPA application that can be represented as DAG of operators that could be executed in a network of computational resources. In this respect, we highlighted the main optimization techniques (i.e., graph rewriting

and operators' placement) used to reconfigure a DSPA DAG in order to satisfy the associated QoS objective. We also described how operators of a DSPA DAG can be executed by Edge-Fog-Cloud nodes. Then, we presented both simulating and real prototype frameworks that allow us to experiment with an Edge-Fog-Cloud architecture.

In the next Chapter, we will present the state of the art on scheduling algorithms used to execute DSPA applications in Edge-Fog-Cloud nodes and we will position the contributions of this thesis.

Chapter 3

State of the Art

Contents

This chapter contains 20 pages.

3.1 Introduction

Identifying optimal scheduling solutions of operators across the resource nodes of a network is a tedious task in particular for nodes with heterogeneous resources (e.g., in Edge/Fog layers) which are shared by different DSPA applications over dynamic IoT data streams.

Several approaches have been introduced to schedule operators in order to fulfil the QoS objective of a DSPA application [19]. They differ in the architecture of the execution environment (hierarchical, peer to peer, centralized, etc.), the scheduling objectives (response time, throughput, network resource usage, computational resource usage, energy consumption, etc.), the optimization methodologies (heuristic, meta-heuristic, mathematical optimization, predictive, etc.), whether the scheduling strategy produces a scheduling solution and later it enables

to reschedule the current solution at run-time. For the latter case, existing approaches additionally differ in how rescheduling is triggered at run-time (time-based, proactive approach, reactive approach, etc.). This chapter attempts to survey the related work and is structured according to the aforementioned dimensions. At the end, we position the contributions of our thesis.

3.2 Execution Environments

Initially DSPA applications were designed to run in centralized environments as an extension of a data base management system (DBMS) server capable of generating data streams. Thus, cluster nodes with high computational resources was the prevailing environment for executing DSPA applications specified e.g., in TelegraphCQ [49].

As data streams were generated by multiple sources, operators started to be distributed across the computational resources of a peer-to-peer (P2P) network. In this context, the challenge was how to improve the performance of a DSPA application by minimizing the network resource usage [8, 22, 24, 46, 47, 77, 95, 96]. High performance computing (HPC) nodes have been additionally exploited in [97] providing high computational resources and interconnectivity (i.e., Infiniband with network delay in the order of microsecond) to speedup the execution of a DSPA application.

With the emergence of the Cloud facilitating to scale up or out computational resources, DSPA applications were naturally deployed in the Cloud to benefit from practically unlimited computational and network resources in order to guarantee high availability and performance (i.e. constrained response time, high throughput) [19]. Wireless sensor networks have been also employed to schedule DSPA applications [98]. For example a hierarchical wireless sensor network is used in [99], where computational resources and network bandwidth capacity are progressively increasing as we go from the bottom to the top of the hierarchy.

New challenges emerged with the upcoming of the IoT, where IoT devices are inter-connected through the Internet, interact and continuously generate huge volumes of data as streams, where the latter may require real-time processing. In this context, the Cloud was de facto the right execution environment. However, IoT data streams need to be transferred to the distant Cloud, involving WAN links, in order to be processed there [100–102]. Cloud-based DSPA solutions may involve a high cost in terms of network bandwidth usage and network delays.

Edge/Fog computing is gaining nowadays increasing attention as it enables to schedule DSPA applications at the IoT network edge. Hence, reduced data size is sent to the Cloud, which decreases the network bandwidth usage and network delays. In this setting, works like [103–109] consider the Edge-Fog-Cloud architecture as the execution environment. Each of these works comes with specific network resource organization. For instance, [109] proposes a shareable hierarchical Edge-Fog-Cloud architecture that comes with several layers of Fog nodes between the IoT devices at the Edge and the Cloud node, while [107, 108] introduce only one layer of Fog nodes. Furthermore, [105,106] propose an Edge-Cloud architecture where the Edge contains

Table 3.1: Execution environments of DSPA applications

	Network architecture			**Layer of hosting nodes**					
	Central.	Hierarc.	P2P	Cluster	HPC	Cloud	Fog	Edge	**Shared**
[49]	✓			✓					
[22, 46, 47, 77, 95]			✓	✓					
[97]			✓		✓				
[100–102]	✓					✓			
[99]		✓						✓	
[98]			✓					✓	
[23, 110–112]		✓				✓		✓	
[114–116]		✓	✓			✓		✓	
[107, 108]		✓	✓			✓	✓	✓	
[109]		✓	✓			✓	✓	✓	✓
[103–106]			✓			✓	✓	✓	
[21, 43]			✓				✓		
Our work		✓				✓	✓	✓	✓

two layers, namely IoT devices and micro data centers, both providing computational resource capacity for hosting operators. In the Cloud layer, they consider federated Cloud sites.

Similarly, works like [23, 110–113] consider a hierarchical resource network where IoT devices are placed at the bottom (Edge) and the Cloud is placed at the top of the hierarchy. A hybrid network architecture is proposed in [114, 114, 115] including a P2P local area network (LAN) for connecting different Edge nodes grouped by region and a hierarchical WAN between the grouped Edge nodes and the Cloud nodes. Finally, distributing DSPA applications only in the Fog layer is considered by [21, 43]. In this setting, Fog nodes are interconnected via a P2P network architecture.

3.2.1 Summarizing table

Table 3.1 summarizes and classifies the execution environments used for scheduling DSPA applications in terms of network architecture to interconnect distributed resource nodes, which can be Centralized (Central), Hierarchical (Hierarc) or Peer-to-Peer (P2P). Then, the type of resource nodes on which operators of DSPA applications can be distributed to, which can be Cluster (or bare-metal), High performance computing (HPC), Cloud, Fog, Edge nodes. Furthermore, whether the target execution environment assumes that its resources can be shared among several applications. The summary shows that in this thesis, we consider a hierarchically Edge-Fog-Cloud architecture as the execution environment, where the resources can be shared among several (DSPA) applications.

3.3 Scheduling Objectives

A scheduling strategy can come with a single or multi-fold objective that specifies the QoS required by a DSPA application. For the latter case, preferences must be set to sort eventually competing scheduling targets that cannot be fulfilled at the same time [22]. To set a common ground for comparison with the contributions of this thesis, in the sequel we distinguish between resource aware and performance aware scheduling objectives.

3.3.1 Performance-aware

Throughput and *Response time* are the two dominant metrics to measure the performance of a DSPA application from an end-user perspective [19], while other metrics used are: application availability [22], accuracy of analytic results [8], data loss [117], etc.

Response time is defined as the time interval between the moment a data item is produced and the moment this data item has been fully processed by a DSPA application [118]. Some other works consider the response time metric as the maximum end-to-end latency [57] or makespan [23]. In the following, we use these terms interchangeably.

In this respect, response time has been considered in [8, 22, 24, 95, 96] that model the problem of scheduling DSPA application as a multi-objective optimization problem on peer network resources. A similar multi-objective optimization is followed by [21] to schedule a DSPA application at the Fog layer by assuming a P2P network. A single objective optimization problem for minimizing the response time of DSPA applications executed over the Edge/Fog and Cloud nodes of P2P network is presented in [103]. Indirect minimization of the makespan is studied in [23] for splitting a DSPA application between the Edge and Cloud aiming actually to minimize the network resource usage to reach the Cloud. The works proposed by [107, 108] consider the response time as objective metric for scheduling a DSPA application across the Edge-Fog-Cloud nodes of hybrid network resource architecture (hierarchical and P2P). [114] minimizes the response time of DSPA application by splitting the resulting application graph model, and distributing the operators across federated Cloud resources and distributed Edge resources.

On the other hand, throughput is defined as the number of data items in a stream that a DSPA application can process in a given amount of time. Throughput is used as a scheduling objective metric in [43] to maintain the maximum number of simultaneously processed data items when scheduling a DSPA application on peer network resources of Fog nodes. This work as well as [119] rather consider the Maximum Sustainable Throughput instead of simple throughput as performance-aware scheduling objective. Going one step beyond, [77] guarantees not only a high throughput but also low response time of DSPA applications running on P2P network nodes with high computational resources such as cluster. In [112] throughput is maximised in the context of mobile cloud computing. A model is proposed in [120] for estimating the throughput of a DSPA application deployed at the Fog based on the placement solution of its operators.

Some other works consider both two metrics as optimization objectives. For instance, [106] solves the operator placement problem between the Cloud and the Edge with the objective to minimize the response time of DSPA application while satisfying a given throughput constraint.

Moreover, [104] attempts to maximise the number of successfully deployed DSPA applications between the Cloud and Fog nodes requiring that network and computational resources should be allocated with parsimony to meet response time constraint for each deployed DSPA applications.

3.3.2 Resource-aware

Resource-aware scheduling aims to consume less *computational resources* (e.g. CPU/GPU, RAM, etc.) and *network resources* (e.g. bandwidth, network delay, etc.) in order to achieve the required QoS of a DSPA application [44]. In this respect, [22, 24, 95, 96] consider minimizing the network resource usage besides the response time while maximizing the application availability when scheduling DSPA applications across a P2P network subject to constraints on the bandwidth usage of network links and the computational resource usage of nodes. Similarly, [74, 121] consider the network resource usage as the scheduling objective to minimize constrained by the response time, the bandwidth usage of network links as well as the computational resource usage. Beside the response time, [21] considers also the network resource usage in their multi-objecting scheduling problem of DSPA application at the Fog. [99] minimizes the combined usage cost of the computational and network resources for deploying a DSPA application over as hierarchical network of sensors.

Besides the objective of minimizing the response time, [114] considers also the constraints of using the computational resources and network bandwidth resources for distributing operators across federated Cloud resources and distributed Edge resources. However, network resource usage is considered as a scheduling objective to minimize in [107, 108] in order to deploy a DSPA application over a hybrid resource network architecture. In the same setting, [8] minimizes the response time as the scheduling objective subject to constraints regarding the usage of the computational and network bandwidth resources.

The *energy* consumed is finally an optimization target when scheduling DSPA applications over limited computational and network resources. In this respect, [110, 111] consider as the scheduling objective, the total energy consumption of mobile devices (e.g. smartphones) processing DSPA application in the continuum of mobile devices and a central data center (e.g. Cloud).

Reducing computational and network resource usage along with energy consumption participate in reducing the *monetary cost* for executing a DSPA application. For instance, [78] minimizes the monetary cost subject to response time constraint for scheduling a DSPA application on peer to peer network resources. Furthermore, besides the response time scheduling objective, [106] considers also minimization of the combined usage cost of computational (i.e.,

cpu and memory) and network resources (i.e., bandwidth), from a monetary cost perspective.

3.4 Scheduling Strategies

So far, we have described the execution environments and the scheduling objectives of DSPA applications. However we have not yet explained how the related works achieve these objectives when scheduling DSPA applications on a specific execution environment. In this subsection, we distinguish scheduling strategies between those proposing an initial (static) operator scheduling solution and those enabling to dynamically adapt the initial scheduling solution in response to the evolution of the DSPA application workload.

3.4.1 Static Scheduling of operators

Scheduling strategies in the related work rely either on mathematical optimization techniques or on heuristic techniques. Mathematical optimization techniques [22, 96, 103] are used to find the minimal or maximal value of an objective function with respect to a set of constraints [122]. To solve the formulated optimization problem, optimization tools (or solvers) may be employed, such as CPLEX [123], Gurobi [124], Gecode [125], Coin and Branch-and-Cut (CBC) [126], etc. On the other hand, heuristic techniques aim to find more rapidly a solution that approximates sufficiently well the optimal solution [23, 24, 78, 110, 111].

3.4.1.1 Mathematical optimization based

[22] relies on integer linear programming (ILP) to formulate the problem of scheduling operators over a P2P network of heterogeneous computational nodes with the objective to minimize the response time and related network resource usage cost as well as to maximize the application availability. This multi-objective optimization problem is reduced to a single objective optimization problem by using simple additive weighting technique. Then, the formulated ILP problem is solved using the optimization software tool CPLEX. This work is extended in [96] in order to replicate the operators then placing them across a P2P network of heterogeneous computational nodes.

Constraint programming (CP) model is also used to find optimal solution to the operator scheduling problem defined as a constraint satisfaction problem. For instance, [103] proposes a framework that models the initial operator scheduling problem between the Cloud and the Edge as a constraint satisfaction problem. The objective is to minimize the response time of DSPA application by minimizing the sum of each individual time for processing an operator on Cloud or Edge nodes and for sending data stream between two connected nodes. The problem considers the resource usage constraints in terms of CPU usage, network bandwidth usage and energy usage as well as the operator replicability constraint. The latter constraint considers the fact that some operators can be deployed on the Edge nodes while some other requiring complex

processing capacity are forced to run only in the Cloud which provides higher computational resource capacity. To solve the formulated CP model, authors use the open-source software optimization tool Gecode.

Mixed Integer Linear Programming (MILP) model is used in [106] to solve the operator placement and replication problem between the IoT devices and Micro Data Center considered as Edge resources and a federation of Cloud sites. The objective is to minimize at the same time, the DSPA application response time and the combined usage costs of computational and network resources. While considering the computational resource usage constraint in terms of CPU and memory, the operator throughput constraint and the placement constraint of the source and sink of the DSPA application. They use the CPLEX tool to solve the resulting MILP model.

Dynamic programming algorithm is used by [113] to partition an IoT application modelled as DAG of operators between the Edge and the Cloud. In this work a specific IoT application is considered that process stream of video in data chunk of equal size. The objective is to minimize the overall completion time (response time) of the graph of operators. In this respect, the proposed dynamic programming algorithm navigates through various possible operators' placement and chooses the one with the minimum total cost w.r.t. the operator graph completion time.

3.4.1.2 Heuristic based

Several heuristic methods have been proposed to approximate an optimal value (min or max) of an objective function with respect to a set of constraints. For instance, [24] proposes several heuristics to solve the operator placement problem on peer resource networks introduced in [22]. These contributions are distinguished between model free and model based heuristics. The proposed heuristics involve the selection of suitable computational resources and/or network links to guide the operator placement decision. In this respect, a penalty function is used for capturing the cost in terms of the objective function of using any computational resource or network link.

Model free heuristics from [24] rely on well known search algorithms such as greedy first fit or local search to solve the operator placement problem. These algorithms involve a risk of getting stuck in a local optimum. To tackle this, authors use Tabu search (TS) algorithm. starting from an initial operator placement, TS finds a local optimum through a set of iterations, then it explores the search space by selecting the best non improving operator placement which can be found in the neighbourhood of the local optimum. This solution uses a limited tabu list to avoid cycling back to an already visited operator placement.

On the other hand, model based heuristics from [24] attempt to reduce the search space by restricting the set of candidate computational resources to host the operator by using a penalty function. Then, they model the operator placement problem as an instance of ILP in the reduced

problem space, which can be solved by using the CPLEX tool.

By modelling a DSPA application as a DAG of operators, some heuristics split the resulting graph of operators into several subgraphs where each individual sub-graph will be mapped to a specific computational resource node with the aim of optimizing some criterion of the QoS requirement.

For instance, the edge-cut algorithm is used in [110, 111], where the authors propose the heuristic algorithm called BOSe. BOSe splits the processing load of DSPA applications defined as Continuous queries (CQs) between a central server and several mobile user devices in order to optimize the trade-off between data communication cost and data processing cost. These costs are expressed in terms of energy consumption on mobile devices. The results of the CQs take the form of individual data streams disseminated to the mobile devices over a shared broadcast medium. BOSe relies on a greedy search of edge-cut moves to split the load of CQs.

The edge-cut algorithm is also used by [23], to propose a uniform approach for deploying a DSPA application between the Cloud and the Edge. The objective is to minimize the network resource usage and indirectly minimize the response time for transmitting data stream produced by IoT devices at the Edge toward the Cloud. In this respect, they consider the minimum edge-cut to split the DAG of operators for each data stream produced at the Edge in two disjoint sub-graphs. The sub-graph that includes the sink of the DAG of operators remains on the Cloud. The remaining sub-graph that is connected with the data source is deployed on the devices at the Edge, the operators that are deployed on the device are subject to the constraint of data locality. Despite the limited computational resources at the Edge, this work does not minimize the usage of these resources.

Furthermore, [78] addresses the problem of distributing a DSPA application on an execution environment constituted with resource nodes which have different management policy (e.g. pricing models). The objective is to minimize the overall resource usage cost in terms of monetary cost of the resource nodes hosting the DSPA application while its end-to-end latency bound is respected. By modeling a DSPA application as a DAG of operators, they split this graph into DAG of control units (CU) where each CU is an operator subgraph that can be deploy on a resource node with a specific management policy. In terms of solution they proposed an algorithm that models the problem as an instance of ILP model which is solved with the CBC solver. To address the scalability issue of the initial algorithm,they propose a heuristic based scheduling strategy. This algorithm initially distributes the available end-to-end latency evenly among all CUs of a DSPA application and then a greedy strategy is used to swap the assigned latency from one CU to another if this leads to a reduced monetary cost. The greedy strategy is repeated step-wise until no more improvement of the monetary cost is possible. However, this work does not consider operator replication neither minimizing specifically computational and network resources.

Splitting a DAG of operators between the Cloud and the Edge is also addressed by [127] aiming to minimize the DSPA application response time, the usage of WAN bandwidth between the Edge and the Cloud as well as the monetary cost of exchanging data streams between the Edge and the Cloud. This problem is constrained by the computational resource usage (CPU and memory), however, it does not consider minimization of the computational resources usage. Given that they model the response time of DSPA application by using queuing model, the input data stream rate to an operator should be lower than the service rate of this operator and the network link should not be saturated. In terms of solution they propose a programming model called R-Pulsar, in which they implement an operator placement algorithm that firstly splits the application graph by regions to deploy between the Edge and the Cloud nodes. The Algorithm gives priority to Edge since Cloud is expected to store message for batch processing while Edge nodes may host the actuators. In this respect if the Edge nodes can not meet the operator computational resource usage constraint then the operator is moved in the Cloud that provide practically unlimited computational resources.

[114] extends the work in [128] to model the DSPA application as data-flow graph. Then authors propose heuristic algorithm that splits the application graph between the Edge and Cloud with the objective to minimize the sum of the end-to-end latency of all the individual operator path in the application graph subject to usage constraint of the computational resources of the Edge nodes and the network resources between both the edge nodes and the Edge node to the Cloud node. The proposed heuristic initially creates a deployment sequence which is essentially a list of operators in topological order and the list of the resources that can host these operators. From the deployment sequences, different heuristics are proposed such as AELS which is a greedy strategy that places operators incrementally by evaluating the aggregated operator path end-to-end latency while respecting the computational and network resource constraints. Furthermore, they extend AELS algorithm to account for the region pattern aka AELS+RP algorithm. This strategy handles complex data-flows that contain multiple paths from sources to sinks and hence, it considers the requirement of each individual operator path and gives priority to message according to their destination. Finally they improve AELS+RP algorithm in order to reduce end-to-end latency so that the search can be applied on the Edge resource node regions which provide higher end-to-end operator path latency. This work does not optimize the computational and network resources. Furthermore, this work considers the operator placement.

[105] proposes a heuristic approach to address the scalability issues of the algorithm proposed in [106]. The proposed heuristic algorithm attempts to reduce the search space of the Edge (IoT devices and Micro data centers) nodes or Cloud nodes, by removing the resource nodes that do not satisfy the operator resource demand. Then, based on the reduced search space, the optimization problem is framed using MILP and solved with CPLEX tool.

[109] introduces a heuristic algorithm that takes into account the trade off between the Fog

nodes location and the application latency requirements for scheduling IoT application module across a hierarchical Edge-Fog-Cloud architecture. However, the Fog nodes are set in more than one layer of Fog. Within this hierarchy the computational resource capacity of the Fog node is increasing as we move from the Edge to the Cloud. The proposed heuristic algorithm starts upon request of application arrival, it first generates a set of group of application components sharing the same source. The algorithm calculates the communication impact of each group in the set then it decides to deploy each group of component on the Fog layer with respect to lower the communication impact. This work consider general IoT application that can be modelled as directed graph while in our work we consider DSPA application that can be modelled as DAG of operator. Furthermore, they focus on application latency requirement while in our work we target not only the application latency requirement but also the network and computational resources usage costs.

[104] addresses the problem of scheduling DSPA applications in the form of query requests between Fog and Cloud resources, assuming that computational resources are practically infinite in the Cloud and limited in the Fog. The objective is to maximize the percentage of successfully deployed DSPA applications. In this respect, a DSPA application is successfully deployed if it satisfies the Fog resource constraint, the DSPA application response time constraint and use as less as possible the Fog to Cloud WAN resources. They distinguish between non time-critical DSPA applications that have their sink in the Cloud and DSPA applications with strict latency requirements that have their sink in the Fog. As baseline solutions they propose state of the art strategies such as as FogOnly that deploys the whole DSPA application in the Fog so that Fog computational resource usage is maximised and the Fog to Cloud network resource usage is maximised, AllDC deploys the whole DSPA application in the Cloud so that the Fog computational resource usage is minimized (i.e., zero) and the Fog to Cloud network resource usage is maximised. On the other hand, they propose heuristic algorithms that make interplay placement of DSPA application between the Fog and the Cloud. In particular, they propose Network aware FogGreedy (NAFogGreedy) algorithm that deploys the whole DSPA application in the Cloud if its sink is in the Cloud and deploy the whole DSPA application in the Fog in case it has its sink in the Fog. For the latter case, if the Fog nodes does not have sufficient resources, then some part of DSPA application will be deployed in the Cloud. They also propose Network-Aware Application oriented (NAAO) algorithm that similarly to NAFogGreedy deploys the whole DSPA application in the Cloud if its sink is in the Cloud. However if the DSPA application has its sink in the Fog, NAAO decides to deploy each individual operator path of DSPA application in Cloud or Fog where this operator path achieves to satisfy the response time constraint.

Table 3.2: Strategies producing a static operator scheduling

Works	**Strategy**		**Policy**		**Objective**				**Constraint**		
	Heur	Math	Plac	Repl	Comp	Net	T	TP	CPU, RAM	Bwd	Repl
[103]		✓	✓				✓		✓	✓	✓
[22]		✓	✓			✓	✓		✓	✓	
[96]		✓	✓	✓		✓	✓		✓	✓	
[106]		✓	✓	✓	✓	✓	✓		✓	✓	
[110, 111]	✓		✓			✓	✓				
[23, 127]	✓		✓			✓	✓		✓	✓	✓
[109]	✓		✓				✓		✓	✓	
[114, 128]	✓		✓				✓		✓	✓	
[105]	✓		✓	✓	✓	✓	✓		✓	✓	
[78, 113]	✓	✓	✓				✓				
[24]	✓	✓	✓			✓	✓		✓	✓	
[104]	✓		✓				✓		✓	✓	
Our work	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓

3.4.1.3 Summarizing table

Table 3.2 summarizes scheduling strategies for static deployment of DSPA application and hence it generate an initial operator placement solution. We classify these works in terms of the following: (i) Strategy identifies whether the approach used is heuristic (Heur) based or mathematical optimization (Math) based; (ii) Policy defines the type of the policy used by the scheduling strategy to achieve the optimization goal – we distinguish between operator placement (Plac) policy and operator replication (Repl) policy; (iii) Objective specifies the QoS objective targeted by the scheduling strategy – this can be whether to minimize the computational resource usage (comp), the network resource usage (net), the application response time (T), or the application throughput (TP); and (iv) Constraint captures the constraints imposed by the execution environment or the application QoS objective – we distinguish between computational resource usage (CPU, RAM) constraint, bandwidth usage (Bwd) constraint, and operator replicability constraint (Repl).

The summary shows that in this thesis, we propose heuristic based and mathematical based scheduling strategies for the static deployment of DSPA application. It also shows that the scheduling strategies that we propose use the operator placement and operator replication policies in order to achieve the objective of minimizing the computational and network resource usage costs, ensuring real time response of DSPA application while satisfying the resource usage constraint and the operator replicability constraint.

3.4.2 Dynamic Scheduling of Continuous Operators

Besides mathematical optimization and heuristic techniques used for scheduling operators at runtime, predictive techniques have been also used to predict future application and system metrics, such as, resource utilisation, DSPA application performance, etc. [77, 116, 129–136]. Based on these predictions, DSPA engines are able to anticipate a possible QoS objective degradation and hence apply adequate strategies. In the sequel, we classify dynamic operator scheduling strategies under the above two categories and the predictive based strategy.

3.4.2.1 Mathematical Optimization based

Despite the scalability concerns raised by mathematical optimization techniques, few works rely on them to solve dynamically the operator scheduling problem. For example, [21] employs ILP to schedule operators in the Fog layer. To accommodate dynamic workloads of nodes due to the spatio-temporal evolution of IoT data stream rates, the scheduling problem originally introduced in [22] has been extended with: (i) the enactment cost, which is the cost per second to run an operator on a Fog node; and (ii) the migration cost, which considers the operator size and the data rate for pulling this operator from the Cloud to a Fog node. [119] leverages Constraint Programming to model the maximum sustainable throughput for dynamic operator placement in the Edge over a P2P network with highly heterogeneous network resources. The problem is constrained by the maximum computational resource capacity of each Edge node and the maximum bandwidth capacity of each network link. This model is solved using the software optimization tool CPLEX.

3.4.2.2 Heuristic based

Heuristic based approaches are largely used to enable dynamic scheduling of operators. This is due to the fact that these approaches come with lower execution cost to approximate the optimal solution and they can scale for large problem instances.

In this respect, [46] proposes a heuristic approach called SBON that dynamically places operator on P2P network resources. SBON relies on multidimensional cost model that considers data stream rates and resource availability of computational and network resources. For each computational resource node, SBON encodes the CPU load and the network latency for routing data streams with its peer nodes. SBON operates in a scalable and decentralized manner that allows individual nodes to make dynamic placement decisions with local information. To determine the placement of an operator to a computational resource node, SBON relies on a spring-relaxation algorithm: a link between operators is modeled as a spring, whose extension is mapped to the network latency of the operator link and spring constant is mapped to the data stream rate flowing on this link.

Similarly, [47] proposes a multidimensional cost model to build a Cartesian space where every

physical node has a virtual position called virtual node such that the Cartesian distance between any pair of physical nodes corresponds to the propagation delay between these physical nodes. Each operator then autonomously determines its optimal virtual node in this cartesian space that minimizes its network resource usage, depending on its current position and the data stream rate of its neighbour operators. The selected virtual node is then mapped to the available closest physical node, if the selected physical node is not overloaded after deployment of the additional operator. Otherwise the algorithm excludes this physical node from the search space and assigns the operator to the next nearest physical node. Even though that [47] proposes a decentralized heuristic while SBON is a centralized heuristic approach. However both two solutions do not minimize the computational resource usage as they consider resource nodes with higher computational resource capacity (i.e., cluster). Nevertheless they consider only the constraints on the computational resource usage. Their proposed scheduling algorithms consider only the operator placement as the optimization policy of DSPA application.

Operator placement policies are additionally exploited. A dynamic scheduling strategy for P2P sensor networks is proposed in [121] with the objective to minimize the network resource usage subject to the constraint of the end-to-end latency. The latency model provided by [46] is used to estimate the propagation delay between sensor nodes. However, data transmission delays are omitted and the operator processing time is considered to be negligible for the small data volumes generated by sensors. The proposed strategy solves the dynamic operator placement problem in two phases. The unconstrained optimization phase finds the minimum network resource usage of the whole operator graph across the overall network of physical sensors. If this initial phase satisfies the end-to-end latency constraint then the optimal solution is found. Otherwise, the constraint satisfaction phase is apply that degrades a little as possible the network resource usage in order to satisfy the end-to-end latency constraint. The proposed solution is also used to solve their extended problem in [74] where they account for the fact that data streams produced by sensors are of considerable size. Thus, the end-to-end latency includes not only the propagation delay but also the tuple transmission delay and the tuple processing delay by an operator on nodes. Despite such extension, however they do not minimize the computational resources usage.

[8] combined both mathematical and heuristic optimization techniques to propose dynamic scheduling strategy for DSPA application over a P2P network of Cloud nodes. The proposed solution, called WASP, takes into account that the WAN bandwidth to reach the Cloud is dynamic along with the workload of DSPA application. In this respect, the objective is to minimize the DSPA application response time whatever the change in the WAN bandwidth and DSPA application workload. Then, WASP employs 3 optimization policies by executing firstly the operator placement policy to minimize the DSPA application response time. To do so they model the operator placement problem as ILP model which is solved with optimization tool

Gurobi. If the operator placement policy does not satisfy the constraint on network bandwidth usage or on computational resource usage, WASP applies respectively the operator reordering to change the current plan of a DSPA application into an equivalent one that solves the network bandwidth bottleneck or the operator replication policy by scaling up/down the number of operator replicas to address the computational resource bottleneck. This work mimics in some how, the concept of content delivery network. Whence the resource network optimization of this work considers distributed Cloud resources interconnected through through WAN characterised by dynamic available bandwidth capacities. Furthermore, this work does not optimize the computational resource usage.

[112] proposes a framework for adaptive computation partitioning and multi-tenancy component as a Service for partitioning and executing DSPA application between the Cloud and mobile devices. The objective is to maximize the application throughput while minimizing the cost of using the Cloud resources with the constraint on bandwidth usage to reach the Cloud. The proposed algorithm splits the application designed as DAG between the mobile devices and the Cloud. However, the proposed algorithm is based on the genetic meta-heuristic algorithm that involves high execution cost and require specific parameter tuning enabling the algorithm to quickly converge toward the optimal solution.

Operator replication policy is frequently used to scale (i.e., increase or decrease) the parallelization degree of an operator in order to improve the computational resource usage and consequently the DSPA application response time. For instance, [137] addresses the problem of end-to-end latency violation when improving dynamically the computational resource utilization of DSPA applications. In this respect, they propose a heuristic approach that firstly analyzes whether the system is overloaded or under-loaded. Then, decide to scale-down or scale-up operator replicas to update the operator placement on the hosts that are respectively overloaded or underloaded. In this respect, the operator placement problem is defined as bin packing problem constrained by the CPU, memory and bandwidth capacity of each host. To solve this operator placement problem, they consider a first fit algorithm by sorting the operator to place on host in decreasing order of their computational resource demands. Then, the upstream or downstream of the currently placed operator on a host is preferred during the placement to favor in memory data communication over transferring data on network links.

Furthermore, [43] exploits operator replication and placement policies. To this end they consider a Monitor, Analyze, Plan and Execute (MAPE) loop design pattern to dynamically schedule operators in the Fog layer. The objective is to maximize the overall DSPA application throughput. The MAPE loop pattern relies on the monitored system and application performance metrics. It analyzes these metrics in order to determine when to scale-up or scale-down an operator. In this respect, for the scale-up, a resource node is added with the replica of the operator that experiences the back-pressure as long as the throughput is lower than a maximum

threshold. For the scale-down, the solution iteratively removes the resource node that hosts the replica of the operator that experience lowest throughput by keeping the maximum throughput higher than a certain threshold. To execute the identified operator placement, the selected resource nodes are sorted by latency with their peers. Conversely to this work, in this thesis we consider not only the Fog resources but also the Cloud resources. Then, we aim at optimizing the trade-off between the usage of the Fog computational resources and the usage of the Cloud network resources. Furthermore, in terms of DSPA application performance, we consider the response time rather than the throughput and aim to ensure the real time response constraint.

3.4.2.3 Predictive based

Several techniques are used in this context to proactively schedule DSPA applications in order to prevent a possible degradation in the DSPA application performances or in its resource usage cost. In this respect, Markov decision process (MDP) [116, 129, 130] and Queuing Theory [77, 131, 132] are the most prominent techniques used to model the dynamic scheduling problem of operators. More recently, machine learning (ML) techniques have been also exploited to propose predictive scheduling of DSPA applications across distributed resources [133–136].

In this respect, [116] models the problem of dynamic operator rescheduling between the Edge and Cloud as MDP with the objective of minimizing the DSPA application response time while satisfying the computational resource constraint of the Edge nodes. To solve the model, they use a reinforcement learning (RL) technique by employing algorithms such as Monte-Carlo Tree Search, Temporal-Difference Tree Search and Q-learning. While the previous work targets Edge-Cloud as execution environment, [129] considers a general geo-distributed resource network, similarly to [22]. The authors model the problem of acquiring an optimal rescheduling of operators as an infinite horizon of MDP. Then, they address the problem of operator scheduling on heterogeneous infrastructure while minimizing the response time and reconfiguration overhead as well as satisfying the computational resource usage constraint. Unlike [116], where the proposed solution leverages RL techniques which suffer for slow convergence in particular as the problem size growth, the solution in [129] involves both RL techniques and linear Function Approximation (FA) techniques. Furthermore, [130] exploits markovian arrival processes to address the problem of auto scaling computational resources (CPU) (rather than scaling operators) while minimizing these computational resources subject to the response time constraint. RL is also used by [138], which proposes a hierarchical approaches for self-adapting DSPA applications at run-time to minimize response time under the constraint of computational resource usage. In this respect, the authors exploit Q-learning algorithms that consider different levels of system knowledge: either a model free learning algorithm or a model based approach. The latter exploits the current knowledge or what can be estimated about the system and application at run-time.

[77] proposes dynamic operator scheduling strategies in order to guarantee the DSPA application response time constraint while minimizing resource consumption across a Geo-distributed network of resources that comes with resource-rich homogeneous nodes. They build a latency model based on a G/G/1 queuing model to predict the latency of each individual operator. Then, based on the prediction model, they employ operator replication policy by increasing the number of replica of each individual operators to satisfy the operator latency constraint. The higher the number of operator replica, the lower the operator latency however the higher the resources consumption. Thus, they use gradient descent algorithm to trade-off between satisfying operator latency and minimizing resource consumption.

[132] designs an adaptive operator scheduling algorithm that replicates and places operators on P2P network in the Cloud. The objective is to ensure lower response time. In this respect, they devise a predictive model of response time and throughput by modeling the problem as a Jackson network model, where each operator is modeled as a GI/G/K queuing model where K indicates the number of replica per operator.

ML techniques such as linear regression is used in [133] to propose a strategy, in response to dynamic data stream rates, that dynamically increases or decreases the number of replicas of each individual operator of a DSPA application. This aims to prevent operator congestion, to free unnecessary resource usage (i.e., RAM, CPU, etc.), and hence to yield results with acceptable data loss and minimum response time. On the other hand, an online support vector regression algorithm is used in [136] to improve the accuracy of data load prediction for dynamic resource allocation of DSPA applications. Furthermore, a neural network technique is used in [134] to forecast variations in the input load of operators. It periodically checks if the current provisioning of resources and operator replication need to be scaled-in or scaled-out to accommodate for foreseeable load fluctuations. Recently, a data stream rate prediction model and resource estimation model have been proposed in [135] to satisfy the response time constraint while minimizing the communication cost across a P2P resource network of clusters. They rely on the neural network prediction based on genetic simulated annealing algorithm to predict the pattern of data stream rate in near future of the cluster, then according to the response time, the resource estimation model adjusts the computational resources allocated to the critical operator of the critical operator path of DSPA application modeled as a DAG.

With the exception of [116] considering an Edge-Cloud architecture, most of the predictive based scheduling strategies are used in the context of execution environments providing higher computational resource capacity [20]. The MDP model introduced in [116] has issues of high computational resource demands and may require high execution time in order to converge to the optimal solution. In particular, MDP requires complete knowledge of the system that is not always possible in dynamic environments such as an Edge-Fog-Cloud architecture.

Table 3.3: Strategies for static and dynamic operator scheduling

Authors	**Strategy**			**Policy**		**Objective**				**Constraint**		
	Math	Heur	Pred	Plac	Repl	Comp	Net	T	TP	CPU, RAM	Bwd	Repl
[21]	✓			✓			✓	✓		✓	✓	
[119]	✓			✓					✓	✓	✓	
[8]	✓	✓		✓	✓			✓		✓	✓	
[95]	✓	✓		✓	✓		✓	✓		✓	✓	
[46,47,74, 121]		✓		✓			✓	✓		✓	✓	
[112]		✓		✓				✓	✓	✓	✓	
[137]		✓		✓	✓			✓		✓	✓	
[43]		✓		✓	✓				✓	✓	✓	
[116]			✓	✓				✓		✓	✓	
[129,130]			✓	✓	✓			✓		✓		
[77,132]			✓	✓	✓			✓		✓	✓	
[133]			✓	✓	✓			✓		✓		
[135]			✓	✓		✓	✓					
Thesis		✓		✓	✓	✓	✓	✓		✓	✓	✓

3.4.2.4 Summarizing table

Table 3.3, uses a classification similar to the one of Table 3.2. There is only an additional Strategy category (Pred) for the predictive based approaches.

In this respect, the summary shows that in this thesis, we propose only heuristic based scheduling strategies for the dynamic deployment of DSPA application. In this context, the dynamic scheduling strategy use the operator placement and operator replication policies in order to achieve the objective of minimizing the computational and network resource usage costs, ensuring real time response of DSPA application while satisfying the resource usage constraint and the operator replicability constraint.

3.4.3 Triggering Dynamic Scheduling of Operators

Proposing dynamic scheduling strategies is necessary for adapting the current operator mapping in response to the evolution of the DSPA application workload or changing conditions of the underlying network and computational resources. However, it is necessary also to decide when these strategies should be triggered. Thus, according to [20] we distinguish three different means used to trigger the adaptation of operator placement at run-time.

Table 3.4: Trigger a dynamic operator scheduling

Works	Time-based	Reactive	Proactive
[139, 141]	✓		
[8, 21, 43, 74, 77, 109, 112, 119, 121, 138, 140]		✓	
[43, 43, 116, 133–136, 138]			✓
Thesis		✓	

3.4.3.1 Time-based

Dynamic scheduling strategies can be performed on a regular basis based on a period of time that can be in the order of few seconds to several minutes. For instance, a time period called activation time is defined in [139] to trigger DS2, a controller for dynamic scaling of distributed operators. DS2 periodically collects operator metrics (i.e., throughput) to build a model that enables to identify whether the running application is overloaded or has over-provisioned resources. Then, at each activation time, DS2 invokes the scheduling strategy for re-scaling the current operator placement.

It is worth noting that triggering a dynamic scheduling of DSPA applications periodically provides design and implementation simplicity as it requires only a single parameter. However the challenge remains in specifying the length of the time period to set that allows collecting sufficient application and system metrics for taking an appropriate scheduling decision. Setting a short period of time involves small quantities of collected metrics, while setting a long period of time may reduce responsiveness to rapidly evolving situations. Thus, it is important to take into account the trade-off between efficiency and responsiveness [20].

3.4.3.2 Reactive

A reactive approach adapts the operator scheduling in reaction to the current system or application changes. Several dynamic operator scheduling strategies have been proposed, such as [8, 21, 74, 77, 109, 121, 138, 140]. The aforementioned strategies are mainly triggered by threshold violations, which are evaluated against the latest collected system or application metrics.

3.4.3.3 Proactive

Proactive approach considers past and current system and application metrics, based on which they try to forecast operator and execution environment changes in the near future, in order to adapt the current operator scheduling in advance if necessary [43, 43, 116, 133–136, 138].

3.4.3.4 Summarizing table

In Table 3.4, we compare the aforementioned works in terms of when dynamic scheduling is triggered: (i) after a fixed time interval (Time-based); (ii) upon a threshold of a monitored

metric is reached (Reactive); and (iii) by predicting (Proactive) in the near future when a threshold of a monitored metric will be reached. This summary shows that this thesis consider a the reactive approach in order to trigger the rescheduling of operator if necessary.

3.5 Discussion and Positioning

The operator scheduling problem aiming to ensure DSPA application performance and at the same time efficient resource usage has been largely discussed in related work. This is demonstrated by the quality and quantity of the contributions made to address the inherent challenges. These contributions differ in the execution environment and related network architecture where the DSPA application is deployed (Edge/Fog, Cloud, Cluster, etc.) (see Table 3.1). They differ also in the scheduling objectives, strategies and policies, as well as the specificities of the DSPA application under consideration (see Table 3.2, Table 3.3 and Table 3.4).

For processing data stream at the IoT network edge, Table 3.1 shows that some works consider only the Edge as the only execution environment [98, 99] and few works consider only the Fog [43]. Some other works consider the execution environment constituted with Edge and Cloud nodes connected by either hierarchical or P2P network [23, 110–112, 114–116]. The works that consider the Edge-Fog-Cloud architecture as the execution environment, the resource nodes in this architecture are connected either only by a P2P network [103–106] or by both P2P and hierarchical networks [107–109] while in this thesis we consider a hierarchical Edge-Fog-Cloud architecture. Furthermore, we consider that the resources across this hierarchical architecture can be shared among several DSPA applications. Few work have dealt with the shareable resource characteristic however at very high level [104, 109].

The computational and network resources across the Edge-Fog-Cloud (or Edge-Cloud) architecture are heterogeneous where the resources at the Edge/Fog can be constrained. Most of the works that rely on this architecture focus on optimizing the network resource usage and the DSPA response time by ensuring only constraint of computational resource usage [23, 104, 109, 115, 142]. While we introduce a resource usage cost model that: (i) addresses both computational and network resources and enables to deal with the trade-offs that are inherent to their joint usage; (ii) characterizes the usage cost of resources by distinguishing between abundant and constrained resources as well as based on their dynamic availability, hence covering both dedicated and shareable resources.

In this respect, we formulate and solve the problem of scheduling operators of DSPA applications across a hierarchical Edge-Fog-Cloud resource architecture that: (i) jointly optimizes the resource usage cost for computational and network resources. Few works take computational resources into account in their optimization goals [105, 120]; (ii) subject to a response time constraint. Few works deal with such a constraint [104], most works that aim to schedule DSPA applications include the response time in their optimization goals (See Table 3.2 and Table 3.3).

Our objective is to schedule a DSPA application in a way that it uses available resources in the most efficient way. This enables saving valuable resources for other DSPA (or non DSPA) applications that share the same resource architecture.

We introduce several scheduling algorithms that deal with different versions of the problem: static scheduling and time aware scheduling (see Table 3.2) and dynamic scheduling (see Table 3.3). In this respect, we consider both mathematical optimization based approach and heuristic based approach. However as the former bears the issues of scalability and high execution cost which is not acceptable for dynamic scheduling strategy of DSPA application with response time constraint. Therefore, we consider the mathematical based optimization approach as benchmark solution for the static scheduling algorithms. Then for the dynamic scheduling strategy, we rely only on heuristic based approach as it enables to approximate the optimal solution within reasonable amount time and take at bay the scalability issues in case of high problem size. The proposed heuristics for dynamic scheduling is triggered through a reactive technique (See Table 3.4).

We extensively and comparatively evaluate our algorithms against several baselines that either we introduce or they originate / are inspired from the state of the art literature [22, 74, 104, 143]. It worth noting that, the related works rely on testbed (real environment) [23, 95] for evalutaing scheduling algorithms. A testbed uses real data on real (or near real) execution environment. For example, [23] uses the Grid'5000 as the Edge-Cloud architecture by deploying the DSPA engine Apache Edgent at the Edge and Apache Flink DSPA engine on the Cloud. It worth noting that in the context of Edge/Fog computing, there is lack of solid established edge-oriented DSPA engines [20]. Furthermore, existing testbeds does not enable to reproduce experiments. For these reasons, we use the simulation tool iFogSim to evaluate the scheduling strategies proposed in this thesis. iFogSim enables to realistically design an execution environment such as Edge-Fog-Cloud architecture, the DSPA application and implement all the scheduling strategies. Furthermore, iFogSim provides means to monitor the execution of a running DSPA application (see Table 2.3.4).

Chapter 4

Resource Usage and Response Time Models

Contents

This chapter contains 15 pages.

4.1 Introduction

In this chapter we highlight the main cost factors when executing DSPA applications over Edge-Fog-Cloud nodes offering network and computational resources. We focus here on cost in terms of required resource quantities, as: (i) resources may be limited; (ii) they may be shareable among multiple DSPA applications; and (iii) their use may incur a monetary cost, which however is not considered here. Long lasting DSPA applications may exhibit a dynamic workload due to the spatio-temporal dynamics of IoT devices generating the data streams. Such dynamic workload impacts not only the resource usage cost of Edge-Fog-Cloud nodes but also the way DSPA

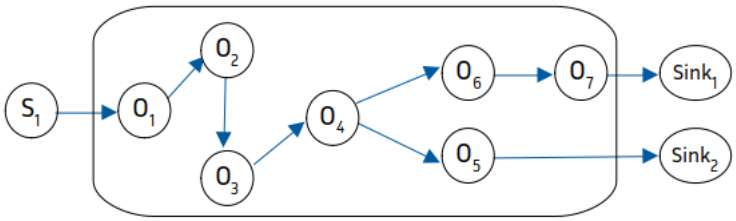


Figure 4.1: DSPA application abstracted as direct acyclic graph

applications should be scheduled in order to ensure a low processing time. In this respect, we introduce also the response time model of DSPA applications.

4.2 Preliminaries

Prior to introducing the resource usage cost model and the response time model, in this section we present core models of the DSPA applications and Edge-Fog-Cloud architectures which are summarized respectively in Table 4.1 and Table 4.2.

4.2.1 DSPA application

We represent a DSPA application as a directed acyclic graph (DAG) of operators (or simply application graph), denoted by G, where the vertices represent operators ($O_x, x \in \mathcal{N}$) and the edges represent the data stream flowing between two operators [7]. 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 sinks ($Sink_1$, $Sink_2$, etc.) that capture the stream of the computed results. Figure 4.1 illustrates this abstraction of 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. Thus, the application graph G is characterized by the following parameters:

Operator *selectivity* (sel_x) In related work [110, 111, 114], the authors often distinguish on one hand selectivity as the ratio between the input and output data tuple size, and on the other hand productivity as the ratio between the input and output data tuple number. However, in this work we consider the operator selectivity as the product of the two metrics. Thus, we define 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.

Edge data rate ($\lambda_{x,y}$) defined as the rate of data stream flow between the two operators connected via this edge.

Operator *cost* (c_x) defined in terms of CPU demand (e.g., million instructions (MI) per byte of data) and/or RAM demand (e.g., Mb per Mb of data) for an operator O_x to process its data load D_x [144].

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

$$D_x = \sum_{i=1}^{I} \omega_x \cdot \lambda_{i,x} \tag{4.1}$$

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., CPU/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:

$$req_x = D_x \cdot c_x \tag{4.2}$$

To replicate and migrate a part of G on different computational resources $n_i = E_i|F_j|C$ of Edge-Fog-Cloud nodes that make part of the resource architecture H, we need to partition G in disjoint sub-graphs, denoted by $Gmig_i$, according to some workload criteria, such that the resulting graph to deploy is defined as follows:

$$G_{dep} = \bigcup_{\forall_{n_i \in H}} Gmig_i \tag{4.3}$$

To specify a replication and migration point in G, we rely on the *edge-cut* algorithm [145] 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. Finally, let the *minimum edge-cut* be the edge-cut that has the smallest value among all the edge-cuts in the application graph G. To calculate the minimum edge-cut we may rely on the Edmond-Karp algorithm [145] that is proven to be efficient for dense graph.

Finally, we consider that each sub-graph $Gmig_i$ to be deployed at the IoT network edge should satisfy the operator replicability constraint to ensure geographical placement constraint of part of DSPA application.

Table 4.1: List of symbols used to model DSPA application

Symbol	Description
G	Directed acyclic graph of operator (application graph)
O_x	Operator $O_x \in G$
e_{xy}	Edge between operators O_x and O_y
S_j	Raw data stream
$\|S_j\|$	Rate of the data stream S_j
w_x	Window size of operator O_x
sel_x	Selectivity of operator O_x
$csel_x$	Cumulated selectivity up to operator O_x
λ_{xy}	Data rate flowing on the edge e_{yx}
c_x	Cost of an operator O_x to process its data load
D_x	Data load of the operator O_x
req_x	Computational resource demand (e.g. CPU/memory) of an operator O_x
$Gmig_i$	Subgraph of the application graph G to place on a resource node n_i
G_{dep}	Union of subgraphs $Gmig_i$ to deploy across the Edge-Fog-Cloud architecture
ec_j	An edge-cut in G, the replication and migration point of S_j
$\|ec_j\|$	Data rate of the edge-cut ec_j

4.2.2 Edge-Fog-Cloud architecture

We abstract an Edge-Fog-Cloud architecture as a hierarchical wide-area resource network defined by the set H=$\{E, F, C\}$ [108]. 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 sum 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 at time t, $M = \sum_{j=1}^{N} m_j(t)$. Figure 4.2 illustrates this abstraction of the Edge-Fog-Cloud architecture.

In this architecture, we distinguish between the computational resources of the Edge/Fog/-Cloud nodes in terms of CPU/GPU or 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 transmitting from an operator to another.

4.2.2.1 Computational resources

For each individual Edge-Fog-Cloud node, we consider the maximum computational resources (in terms of CPU/GPU, RAM) 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. We additionally consider the available computational resources 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

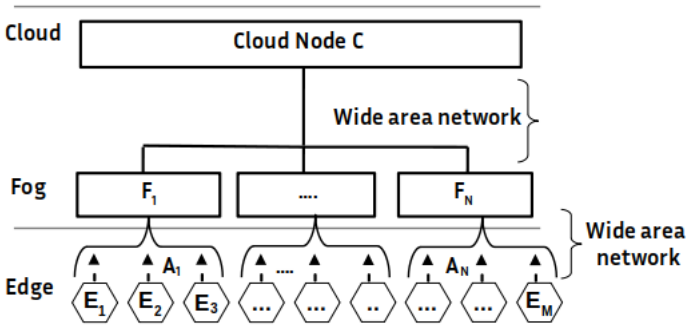


Figure 4.2: Hierarchical Edge-Fog-Cloud Archietcture

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, 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 or non DSPA) applications.

4.2.2.2 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 [75]. 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 and processes. 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 utilization of other (DSPA or non DSPA) applications, even if nb_{F_jC} is not shared

Table 4.2: List of symbols used to model the Edge-Fog-Cloud architecture

Symbol	Description
H	Hierarchical wide area resource network modelling Edge-Fog-Cloud architecture
E_i, F_j, C	Resource node at respectively Edge layer, Fog layer and Cloud layer
n_i,	Abstracts an Edge node E_i, a Fog node F_j or a Cloud node C (i.e. $n_i = E_i \vert F_j \vert C$)
A_j	Geographical area of IoT devices
m_j	Number of IoT devices per geographical area A_j
cm_{n_i}	Maximum computational resource capacity of node n_i
cma_{n_i}	Available computational resource capacity of node n_i
cmu_{n_i}	Demand of computational resource usage on a node n_i
$nl_{n_i n_j}$	Network link for node $n_i = E_i \vert F_j$ to node $n_j = F_j \vert C$
$nb_{n_i n_j}$	Maximum network bandwidth capacity on the network link $nl_{n_i n_j}$
$nba_{n_i n_j}$	Available network bandwidth capacity on the network link $nl_{n_i n_j}$
$nbu_{n_i n_j}$	Demand of network bandwidth usage on the network link $nl_{n_i n_j}$

with them.

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 $nd_{E_i F_j}$ be the network delays of the network link from an Edge node E_i to its nearest Fog node F_j and $nd_{F_j C}$ the network delay of the network link from a Fog node F_j to the Cloud node C. In Chapter 6, we discuss the network delay in detail.

4.3 Resource usage model

Computational and network resources across the Edge-Fog-Cloud architecture are heterogeneous as they can be constrained and this in very different degrees. Furthermore, these resources are in most cases shareable among several (DSPA) applications. So, it is important to assess the usage cost of these resources in a representative way that can ensure their most efficient utilization for whether at static (initial) deployment or dynamic deployment of DSPA applications.

In essence when we deploy statically a DSPA application, we do not take into account its current deployment state neither the current state of the Edge-Fog-Cloud resources on which it is deployed. However, in the case of dynamic deployment, we need to take into account, the actual deployment state of the DSPA application as well as the actual Edge-Fog-Cloud resources. One step further, we need also to take into account the state of the resource if it is selected to be used.

In this respect, prior to introduce the resource usage cost model, we first introduce how to

weight the usage of each resource of the Edge-Fog-Cloud architecture. Table 4.3 summarizes the symbol used for modelling the resource usage cost.

4.3.1 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 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 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. It 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.

To weight a resource, we distinguish between weight factor that is calculated statically in the case of static deployment of DSPA application and the weight factor that is calculated dynamically in case dynamic deployment of DSPA applications.

4.3.1.1 Static weights

This weight factor should be used in the case a DSPA application is deployed statically from scratch across the Edge-Fog-Cloud architecture. In this context, it only considers the maximum reserved capacities of each individual resources of the Edge-Fog-Cloud architecture.

In particular, this weight factor reflects the strategy that we should use with parsimony resources with constrained maximum reserved capacities and favor the usage of resources with higher maximum reserved capacities. In this respect, we weight the request Req of using each resources across the Edge-Fog-Cloud architecture by the inverse of its maximum reserved capacity Max. Hence, the weight factor in this case is formulated as following:

$$W = \frac{1}{Max} \tag{4.4}$$

4.3.1.2 Dynamic weights

This weight factor should be used in the case of a DSPA application is deployed (redeployed) dynamically at run-time across the Edge-Fog-Cloud architecture. In this context, we need to take into account the current state (available capacity) of each individual resources across the Edge-Fog-Cloud architecture which can already be used by the current DSPA application or other (DSPA) application.

Specifically, this dynamic weight factor reflects the strategy that, before selecting a resource, we should take into account not only its maximum capacity but also its current state and even further its resulting state if this resource is selected to be used.

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: $Max - Avail$. The current usage ratio of this resource is $\frac{(Max-Avail)}{Max}$. If we want to deploy a DSPA application that requires resource usage Req on this resource, the usage ratio will be $\frac{(Max-Avail+Req)}{Max}$. We propose to use this resulting usage ratio to weight the usage cost of a resource when minimizing the resource usage cost. Hence, when deploying a new DSPA application, we will favor the usage of resources with low resulting usage ratios.

$$W = \frac{(Max - Avail + Req)}{Max} \equiv \frac{Max}{Max} - \frac{Avail}{Max} + \frac{Req}{Max} \equiv 1 - \frac{Avail}{Max} + \frac{Req}{Max} \tag{4.5}$$

Another intuitive interpretation of W can be drawn from Formula 4.5 as following: 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 Edge-Fog-Cloud 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 efficiently use the node or network link that has the smallest available resource capacity.

4.3.2 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 either of the weight versions.

For the case of the static weight, we use Formula (4.4) and hence we calculate the weight W_{E_i}, W_{F_j} and W_C of using respectively an Edge node E_i, a Fog node F_j and Cloud node C as following:

$$W_{E_i} = \frac{1}{cm_{E_i}} \tag{4.6}$$

$$W_{F_j} = \frac{1}{cm_{F_j}} \tag{4.7}$$

$$W_C = \frac{1}{cm_C} \tag{4.8}$$

However for the case of the dynamic weight, we use Formula (4.5). In this respect, we calculate the weight W_{E_i}, W_{F_j} and W_C of using respectively an Edge node E_i, a Fog node F_j and Cloud node C as following:

$$W_{E_i} = 1 - \frac{cma_{E_i}}{cm_{E_i}} + \frac{cmu_{E_i}}{cm_{E_i}} \tag{4.9}$$

$$W_{F_j} = 1 - \frac{cma_{F_j}}{cm_{F_j}} + \frac{cmu_{F_j}}{cm_{F_j}} \tag{4.10}$$

$$W_C = 1 - \frac{cma_C}{cm_C} + \frac{cmu_C}{cm_C} \tag{4.11}$$

Given the above Formulas that weight (statically or 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:

$$cru = \sum_{i=1}^{M}(cmu_{E_i} * W_{E_i}) + \sum_{j=1}^{N}(cmu_{F_j} * W_{F_j}) + (cmu_C * W_C) \tag{4.12}$$

Where 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 C. Moreover W_{E_i}, W_{F_j} and W_C can be either of the weight versions and one should never use different weight version in the same model.

4.3.3 Network resource usage cost

Given that we abstracted the Edge-Fog-Cloud architecture as a *hierarchical WAN resources*, we observe that the network bandwidth increases up in the hierarchy as we go from the Edge to the Cloud and the network delay also increases as we go from the Edge to the Cloud. Furthermore, 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 [8].

In the literature [22, 46], 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. In this respect, the cost of using a network link is calculated by multiplying the requested network bandwidth usage by the weight factor (static or dynamic) of using this link and its network delay.

Table 4.3: List of symbols used for resource usage cost model

Symbol	Description
W	Weight of using a resource (network or computational)
Max	Maximum reserved resource capacity (network or computational)
$Avail$	Available resource capacity (network or computational)
Req	Request of using a resource (network or computational)
cru_{n_i}	Computational resource usage cost on node node n_i
cru	Overall computational resource usage cost across Edge-Fog-Cloud architecture
$nru_{n_i n_j}$	Network resource usage cost of network link $nl_{n_i n_j}$
nru	Overall network resource usage cost of all the network links in Edge-Fog-Cloud

To calculate statically 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 4.4 as following:

$$W_{E_iF_j} = \frac{1}{nb_{E_iF_j}} \tag{4.13}$$

$$W_{F_jC} = \frac{1}{nb_{F_jC}} \tag{4.14}$$

On other hand, to calculate dynamically 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 4.5 as following:

$$W_{E_iF_j} = 1 - \frac{nba_{E_iF_j}}{nb_{E_iF_j}} + \frac{nbu_{E_iF_j}}{nb_{E_iF_j}} \tag{4.15}$$

$$W_{F_jC} = 1 - \frac{nba_{F_jC}}{nb_{F_jC}} + \frac{nbu_{F_jC}}{nb_{F_jC}} \tag{4.16}$$

Given the weight factor statically or dynamically calculated, the overall network resource usage cost is formulated as following:

$$nru = \sum_{j=1}^{N}\sum_{i=1}^{M}(nbu_{E_iF_j} \cdot W_{E_iF_j} \cdot nd_{E_iF_j}) + \sum_{j=1}^{N}(nbu_{F_jC} \cdot W_{F_jC} \cdot nd_{F_jC}) \tag{4.17}$$

Where M is the total number of the Edge nodes E_i and N is the total number of the Fog nodes F_j.

4.4 Response time model

According to the criteria of minimizing cru and nru, the resulting G_{dep} defined in Formula (4.3) becomes the disjoint partition in subgraphs $Gmig_i$ to deploy across the Edge-Fog-Cloud

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 [22], we define the response time T as the worst end-to-end latency $L\pi_{ij}$ among all the operator paths $\pi_{ij} \in G_{dep}$:

$$T = \max_{\pi_{ij} \in G_{dep}} (L\pi_{ij}) \tag{4.18}$$

where each data stream S_j is processed by $n_\pi > 0$ operator paths π_{ij} in G_{dep} with i=$\{1, \ldots, n_\pi\}$.

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

$$L\pi_{ij} = \sum_{e_{xy} \in \pi_{ij}} nd_{\mathcal{M}(x),\mathcal{M}(y)} + \sum_{O_x \in \pi_{ij}} l_x \tag{4.19}$$

where $\mathcal{M}$ 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.

In the following, we define in detail the principle components that participate in the response time model, namely the network delay and the operator latency.

4.4.1 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 [75]:

$$nd_{n_i n_j} = pd_{n_i n_j} + td_{n_i n_j} \tag{4.20}$$

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.

4.4.1.1 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 [74]. 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 [146].

4.4.1.2 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. Several techniques have been proposed for this purpose [75].

Therefore in this thesis 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). Then, the transmission delay of any data d_{xy} is calculated as follows [74]:

$$td_{n_i n_j} = td'_{n_i n_j} \cdot \frac{|d_{xy}|}{|d|} \tag{4.21}$$

4.4.2 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 [127] and it is formulated as following:

$$\mu_x = \frac{cpu_j}{req_x} \tag{4.22}$$

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 which is defined in Formula (4.2).

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 [144]. Thus, if $MIPS_j$ is the total CPU capacity of a resource node n_j, we calculate cpu_j as follows [74]:

$$cpu_j = min(MIPS_j, \frac{MIPS_j}{q_j}) \tag{4.23}$$

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 with one server and following the first in first out policy [147] as depicted in Figure 4.3. Then, the operator latency l_x is approximated as follows:

$$E(l_x) = E(wt_x) + \frac{1}{\mu_x} \tag{4.24}$$

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.

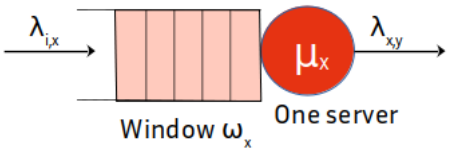


Figure 4.3: Modeling operator as a queuing system

4.4.2.1 Time based sliding window

This type of window is characterized by the temporal extend of the window, called window time ω_x, and the progression temporal step, called sliding time β_x where $\omega_x > \beta_x$ [37]. 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 [148].

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:

$$E(wt_x) = \frac{1}{\mu_x} \cdot \frac{\gamma}{(1-\gamma)} \tag{4.25}$$

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 [149].

4.4.2.2 Tuple 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 [37]. 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:

$$E(wt_x) = \frac{\rho_x}{2 \cdot \mu_x \cdot (1-\rho_x)} \tag{4.26}$$

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

Table 4.4: List of symbols used for the response time model

Symbol	Description
$nd_{n_i n_j}$	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 i for processing data stream S_j
T	DSPA Application response time
$\mathcal{M}$	Mapping function that gives the host resource node of an operator
l_x	Latency of an operator O_x for processing its data load
μ_x	Service rate of an operator O_x for processing its data load
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 items in the queue of an operator O_x
ω_x	Window size of an operator O_x
β_x	Sliding size of an operator O_x
$MIPS_j$	Maximum CPU capacity of a resource node n_j (in terms of MIPS)
cpu_j	Allocated CPU resources to each process on a resource node n_j (in terms of MIPS)
q_j	Total number of processes running on resource node n_j

4.5 Conclusion

In this chapter, we modelled a DSPA application as DAG of operators and the Edge-Fog-Cloud architecture as hierarchical WAN resources. We then presented a general cost model that weights statically or dynamically the usage of a resource when distributing operators of DSPA application across the Edge-Fog-Cloud architecture.

In this respect, the static weight should favor the usage of resources with high maximum capacities. The resources with constrained maximum capacities should be used efficiently. On the other hand, the dynamic weight should favor the usage of resources with high available capacities while resources with constrained available capacities should be used with parsimony. Furthermore, we introduce the response time model to take into account the real time response constraint of DSPA application.

In the next chapter, we will exploit the proposed resource usage cost model to devise resource aware scheduling algorithms of DSPA application across the Edge-Fog-Cloud architecture. Then latter we will extend the resource aware scheduling algorithms to take into the response time constraint of DSPA application to be scheduled across the Edge-Fog-Cloud architecture.

Chapter 5

Resource-Aware Scheduling of Continuous Operators

Contents

This chapter contains 28 pages.

5.1 Introduction

This chapter relies on the resource usage cost model presented in Chapter 4 (see Section 4.3). We consider static deployment of the DSPA application across the Edge-Fog-Cloud architecture. Consequently, the DSPA application is deployed from scratch and hence the maximum capacity of a resource is available to the reserving DSPA application. We also consider in this Chapter

that data steam are not processed by IoT devices at the Edge layer and the computational resources of the Cloud are practically unlimited while those of the Fog layer are limited.

Additionally, the algorithms of this chapter that aim at optimizing resource usage calculate a new operator placement each time they are employed without taking into account the previous state of resource usage. Consequently, they assume that the available capacity of a resource equals its maximum capacity.

In this respect, we formulate the resource allocation problem to identify opportunities for operator replication and placement between the Fog and Cloud nodes that optimizes at the same time the computational resource usage cost of Fog nodes and the network resources usage cost to reach the Cloud while respecting the maximum resource usage constraints.

In this perspective, in Section 5.3 we first propose a baseline solution called resource constraint satisfaction (RCS) algorithm that identifies scheduling solution of operator between the Fog and the Cloud nodes while satisfying only the maximum resource usage constraint. In essence, the RCS algorithm improves the typical Cloud based processing architecture where the DSPA application is deployed in the Cloud and the data streams produced by IoT devices are directly sent to the Cloud.

As the RCS algorithm does not optimize the resources usage cost, in Section 5.4 we formulate the operators scheduling problem as a single objective optimization (SOO) problem of the combined usage costs of the computational resources and the network resources and we show that the SOO problem is an NP-hard problem. Then, we propose new scheduling strategies for solving the SOO problem. To do so, we formulate the SOO problem as an instance of integer linear programming (ILP) model subject to inequality constraints and we rely on the CPLEX tool [123] to find an optimal solution of this problem. As our SOO-CPLEX solution may incur a high execution time for large problem sizes, we introduce a heuristic algorithm called SOO-H. SOO-H exploits the characteristics of the application graph G and of the hierarchical network model abstracting the Edge-Fog-Cloud architecture to achieve an optimal overall resource usage cost, both in best-case and worst-case executions.

The contributions described in this Chapter have been published in respectively the fourth International Workshop on Edge Systems, Analytics and Networking (EdgeSys 2021) [150] and in the Sixth International Conference on Fog and Mobile Edge Computing (FMEC 2021) [151].

5.2 Resource allocation problem

We choose to initially work under the assumption of static deployment of DSPA application across the Edge-Fog-Cloud architecture, where the DSPA applications are deployed from scratch without taking into account the current resource state of the Edge-Fog-Cloud architecture. To enable efficient usage of constrained resources that may be selected to host a part of the DSPA application, we use the static weight version (see Formula Formula (4.4)) in which the usage of

each resource is weighted by the inverse of its maximum reserved capacity. In other terms the cost of using an Edge-Fog-Cloud resource is calculated by multiplying the request of using this resource by the inverse of its maximum resource capacity.

5.2.1 Computational resource usage cost

If we consider cmu_{Ei}, cmu_{Fj} and cmu_C respectively as the request of using the Edge node E_i, the Fog node F_j and the Cloud node C that have respectively maximum computational resource capacities cm_{Ei}, cm_{Fj} and cmu_C. Considering the static weight version, the weight of using the Edge node E_i, the Fog node F_j or the Cloud node C is respectively $W_{E_i} = \frac{1}{cm_{E_i}}$, $W_{F_j} = \frac{1}{cm_{F_j}}$ and $W_C = \frac{1}{cm_C}$.

In this thesis, to calculate the computational resource usage cost, we make the following intuitive assumptions [108]: (i) the computational resources are practically unlimited in the Cloud, $cm_C \to \infty$, thanks to the on-demand resource scaling, and hence the weight in the cloud is practically zero, $W_C \to 0$; (ii) the computational resources of Fog nodes are limited as they can not be scaled on demand, thus the weight in a Fog node is non-zero, $W_{F_j} \in]0, 1]$; and (iii) data stream produced by IoT devices are not processed by the Edge nodes. Then, the focus of our work is to minimize the Fog computational resource usage cost. In this respect, the overall computational resource usage cost (i.e. cru) defined in Formula (4.12) becomes:

$$cru = \sum_{j=1}^{N} cmu_{F_j} \cdot W_{F_j} \equiv \sum_{j=1}^{N} cmu_{F_j} \cdot \frac{1}{cm_{F_j}} \tag{5.1}$$

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

5.2.2 Network resource usage cost

For the network resource usage cost, If we consider $nbu_{E_iF_j}$ and nbu_{F_jC} as the request of using the network link respectively from the Edge node E_i to the Fog node F_j and from the the Fog node F_j to the Cloud node C. These network links have respectively maximum network bandwidth capacities $nb_{E_iF_j}$ and nb_{F_jC} and network delays respectively $nd_{E_iF_j}$ and nd_{F_jC}. Considering the static weight version defined in (4.4), the weights of using these network links are respectively $W_{E_iF_j} = \frac{1}{nb_{E_iF_j}}$ and $W_{F_jC} = \frac{1}{nb_{F_jC}}$.

By assuming that data streams are not processed at the Edge, the Edge to Fog network resource usage cost is constant as the data stream produced by IoT devices at the Edge reach the Fog in anyway. Thus, the cost part concerning the Edge to Fog network links is fixed (set as c constant) in the overall network resource cost (i.e. nru) defined in Formula (4.17). We then focus on minimizing the Fog to Cloud network resource usage cost that why Formula (4.17) becomes:

$$nru = c + \sum_{j=1}^{N} nbu_{F_jC} \cdot W_{F_jC} \cdot nd_{F_jC} \equiv c + \sum_{j=1}^{N} nbu_{F_jC} \cdot \frac{1}{nb_{F_jC}} \cdot nd_{F_jC} \tag{5.2}$$

Although the network delay on a WAN link can vary due to the condition of the underlying physical link that is shared by multiple data connections [46], we assume that the network delays nd_{F_jC} between each Fog node and the Cloud can be considered equal and that can be assigned statically calculated average values. Consequently, rather than to consider the maximum network bandwidth capacity of each individual Fog to Cloud network link, we assume nb_{FC} as their aggregated values. If we assume nd_{FC} as the average network delay value of the Fog to Cloud network links, we can consider nd_{FC} as a constant a. Consequently, Formula (5.2) becomes:

$$nru = c + a \cdot \sum_{j=1}^{N} nbu_{F_jC} \cdot \frac{1}{nb_{FC}} \tag{5.3}$$

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

$$B = \sum_{j=1}^{N} nbu_{F_jC} \tag{5.4}$$

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 [12]. Thus, we assume an upper threshold $Bmax$ of B which is set for a specific DSPA application.

5.2.3 Problem statement

Our goal is to schedule operators of the application graph G so that we respect resource constraints and we jointly minimize the computational resource usage cost on the Fog nodes and the Fog-to-Cloud network resource usage cost. The problem is formalized as follows:

$$\text{minimize} \quad (cru, nru) \tag{5.5}$$

$$\text{subject to} \quad B \leq Bmax, \tag{5.6}$$

$$cmu_{F_j} \leq cm_{F_j}\ j = 1, \ldots, N, . \tag{5.7}$$

Equation (5.7) represents the constraint regarding the maximum computational resource usage (i.e., CPU/RAM usage) of each individual Fog nodes, while Equation (5.6) captures the constraint on the maximum Fog to Cloud network bandwidth usage.

Clearly our two optimization objectives are conflicting. On one hand, minimizing the overall usage cost of computational resources in the Fog, i.e., cru, implies to place all operators of G in the Cloud, which results in low (zero) cru and high (overall Fog-to-Cloud) network resource usage cost, i.e., nru. On the other hand, minimizing the overall usage cost of network resources between the Fog and the Cloud, i.e., nru implies to place in the Fog the operators of G delimited by the minimum edge-cut, which may result in high or low cru and low nru. Having two optimization objectives which may be in opposition with each other indicates that an optimal scheduling solution will not be unique. As a matter of fact, a set of optimal solutions, where each represents a different trade-off between the objectives namely the computational vs network resource usage for scheduling DSPA applications between the Fog and the Cloud; the choice among these solutions can be decided by considering different resource allocation policies prioritizing one of the two optimization objectives.

5.3 Resource Constraint Satisfaction (RCS)

We initially propose a baseline solution, which we call Resource Constraint Satisfaction (RCS) algorithm. RCS algorithm regulates dynamically the computational and network resource usage so that the resource usage constraints are continuously satisfied.

$$B \leq Bmax, \tag{5.8}$$

$$cmu_{F_j} \leq cm_{F_j}\ j = 1, \ldots, N, \tag{5.9}$$

$$B \geq Bmin.. \tag{5.10}$$

We additionally introduce the constraint (5.10), where $Bmin$ is a lower threshold of B, to avoid oscillation of operator placement between the Fog and the Cloud. Based on $Bmin$, Fog computational resources are released when their usage is not necessary.

RCS algorithm is a resource aware based scheduling strategy aiming at using as less as possible the Fog computational resources.

5.3.1 RCS algorithm

We assume that the DSPA application is initially deployed in the Cloud as all the data streams produced at the Edge arrive to the Cloud in anyway. In this respect, RCS requires to constantly monitor the Fog-to-Cloud network bandwidth usage and the Fog computational resource usage.

RCS for which the pseudo code is presented in Algorithm 1 proceeds as follows: If the Fog-to-Cloud network bandwidth usage constraint (i.e., $B \leq Bmax$) is not satisfied, RCS triggers

the function replicateAndMigrateToFog() (line 1-2). If the constraint $B \geq Bmin$ is not satisfied, RCS triggers the function migrateBackToCloud() (line 3-4). Finally, if the computational resource usage constraint (i.e., $cmu_{F_j} > cm_{F_j}$) of a Fog node F_j is not satisfied, RCS triggers the function adjustEdgeCut() (line 5-6). In the following, we present in detail these three functions used by the RCS algorithm.

Algorithm 1: RCS

Input: G, application graph
Input: S, set of S_j arriving to the Cloud
Input: B, Fog-to-Cloud network bandwidth usage
Input: $Bmax$, Upper threshold for B
Input: $Bmin$, Lower threshold for B
Input: Grep $\subseteq$ G, replicable subgraph in G
Input: Fog, set of Fog nodes F_j

```
1 if B > Bmax then
2     ReplicateAndMigrateToFog()
3 else if B < Bmin then
4     MigrateBackToCloud()
5 else if cmu_{F_j} > cm_{F_j} then
6     AdjustEdgeCut()
```

5.3.1.1 Replicate and migrate to the Fog

When $B > Bmax$, RCS triggers the function replicateAndMigrateToFog (Algorithm 2). The objective is to lower the Fog-to-Cloud network bandwidth usage B below the upper threshold $Bmax$ while using as less as possible the Fog computational resources. In this respect, RCS favors migrating the processing of high-rate data streams, which have a high impact on the Fog-to-Cloud network bandwidth usage. To this end, RCS sorts all the data streams S_j based on their rates, then RCS selects the highest-rate data stream S_j (lines 2-4). For the selected S_j, RCS identifies the maximum subgraph $Gsat_j \subseteq Grep$ that can be migrated to the nearest Fog node F_j while satisfying the related computational resource constraint (5.7). Then, RCS identifies the subgraph $Gmig_j \subseteq Gsat_j$ delimited by the minimum edge-cut of $Gsat_j$, and marks $Gmig_j$ as the part of the application graph G to replicate and migrate to the Fog (lines 7-12). RCS updates B and checks if B is still above $Bmax$, in order to select the next highest-rate data stream (lines 13-16). Otherwise RCS performs the reconfiguration of G (lines 17-19).

Algorithm 2: RCS::replicateAndMigrateToFog

```
1  Function replicateAndMigrateToFog():
2      Sort S in decreasing order
3      Pick S_j on top of S if not yet migrated on the Fog
4      Selected ← ∅∪S_j
5      M ← ∅, set of subgraphs to deploy on the Fog
6      while Selected ≠ ∅ do
7          Pick S_j on top of Selected
8          Select Fog node F_j receiving S_j
9          Identify Gsat_j ⊆ Grep while cmu_Fj ≤ cm_Fj
10         Find minimum edge-cut ec_j in Gsat_j
11         Find Gmig_j ⊆ Gsat_j delimited by ec_j
12         M[j] ← Gmig_j
13         B ← B − |S_j| + |ec_j|
14         if B > Bmax then
15             Pick S_j on top of S if not yet migrated on the Fog
16             Selected ← Selected ∪S_j
17     Rewrite G_dep to include all Gmig_j ∈ M
18     Deploy the new G_dep
19     Redirect all selected S_j to be processed on the Fog
```

5.3.1.2 Migrate back to the Cloud

When RCS has replicated a part of DSPA application on the Fog, we need a way to move the processing of these data streams back to the Cloud when using the Fog computational resources is not necessary any more.

Thus, when B gets lower than $Bmin$, RCS triggers the function migrateBackToCloud (Algorithm 3). The objective is to raise B as much as possible above the lower threshold $Bmin$ while still satisfying the constraint $B \leq Bmax$. In this respect, RCS favors migrating back to the Cloud the data streams previously migrated to the Fog that will have the lowest impact on the constraint $B \leq Bmax$. To this end, RCS sorts all the data streams S_j migrated to the Fog, then selects the lowest-rate one (lines 2-4). Based on the selected data stream S_j, RCS identifies $Gmig_j$ to be removed from the Fog, and checks if removing $Gmig_j$ will still keep B lower than the upper threshold $Bmax$ (lines 7-10). If this is true, RCS actually updates B, marks $Gmig_j$ to be removed, and selects the next lowest-rate data stream S_j (lines 11-14). Otherwise, RCS performs the reconfiguration of G by removing all marked subgraphs $Gmig_j$ and redirecting the corresponding data streams to be processed directly in the Cloud (lines 15-17).

Algorithm 3: RCS::migrateBackToCloud

```
1  Function migrateBackToCloud():
2      Sort S in increasing order
3      Pick S_j on top of S, if migrated on Fog
4      Selected ← ∅∪S_j
5      R set of Gmig_j to remove on the Fog
6      while Selected ≠ ∅ do
7          Pick S_j on top of Selected
8          ec_j ← MR[j]
9          Gmig_j ← M[j]
10         if (B + |S_j| − |ec_j|) < Bmax then
11             B ← B + |S_j| − |ec_j|
12             RM[j] ← Gmig_j
13             Pick S_j on top of S, if migrated on Fog
14             Selected ← Selected ∪S_j
15     Rewrite G_dep to remove all Gmig_j ∈ R
16     Deploy the new G_dep
17     Redirect all selected S_j to be processed in Cloud
```

5.3.1.3 Adjust edge-cut

When the computational resource usage constraint is not satisfied for at least one of the Fog nodes F_j (i.e., $cmu_{F_j} > cm_{F_j}$) on which a subgraph $Gmig_j$ is deployed, RCS triggers the function adjustEdgeCut (Algorithm 4). The objective is to readjust the current subgraph $Gmig_j$ so that the resulting cmu_{F_j} satisfies the constraint $cmu_{F_j} \leq cm_{F_j}$.

Algorithm 4: RCS::adjustEdgeCut

```
1  Function adjustEdgeCut():
2      Get current S_j served by F_j
3      Gmig_current ← M[j]
4      Get Gsat_j ⊆ Gmig_current where cmu_Fj ≤ cm_Fj
5      Find minimum ec_j in Gsat_j
6      Find Gmig_j ⊆ Gsat_j delimited by ec_j
7      Rewrite G_dep to replace Gmig_current by Gmig_j
8      Deploy the new G_dep
```

In this respect, we identify the data stream S_j that is served by the Fog node F_j; then we identify the subgraph $Gmig_{current}$ that is currently deployed on this Fog node F_j (lines 2-3). Based on $Gmig_{current}$ and the rate of the data stream S_j, we identify the subgraph $Gsat_j \subseteq Gmig_{current}$ that satisfies the constraint $cmu_{F_j} > cm_{F_j}$. Then we select the minimum edge-cut ec_j in $Gsat_j$, based on which we identify the subgraph $Gmig_j \subseteq Gsat_j$ to replicate (lines 4-6). After having identified the new $Gmig_j$, we perform the reconfiguration of the application graph G to replace the current $Gmig_{current}$ by $Gmig_j$ on the Fog node F_j(lines 7-8).

5.4 Single Objective Optimization (SOO)

The RCS algorithm merely aims to satisfy the constraints defined in our resource allocation problem (i.e., Section 5.2.3). Even though RCS uses as less as possible the computational resources in the Fog, it is oblivious to the minimization of the cost of the computational and network resources actually used. In this section, we model the resource allocation problem for scheduling operators between the Fog and Cloud nodes as a Single Objective Optimization (SOO) problem. We show that this SOO problem is NP-hard and we propose adequate solutions namely: (i) SOO-CPLEX, which models the SOO problem as an integer linear programming (ILP) problem and solves it with the CPLEX solver; and (ii) SOO-H, which uses a heuristic approach to solve the SOO problem more time-efficiently.

5.4.1 Problem formulation

We transform the multi-objective optimization problem introduced in Section 5.2.3 to a single-objective optimization (SOO) problem by taking the weighted sum of the two optimization metrics. As *cru* and *nru* have different units and scales, 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 where the usage cost of each Fog node F_j takes value between 0 and 1 (see Formula (5.1)). Hence, if we have N ($N \geq 1$) Fog nodes that may be used as necessary, *cru* will take values between 0 and N. Then, we normalize *cru* to CRU as following:

$$CRU = \frac{cru}{cru_{max}} \tag{5.11}$$

With $cru_{max} = N$, CRU in this case will take values between 0 and 1.

For the overall network resource usage cost (i.e. nru in Formula (5.3)), the weight of using a Fog to Cloud network bandwidth is equal to the inverse of the aggregated maximum bandwidth capacity of each individual Fog to Cloud network link. Hence, *nru* takes values between 0 and 1 if we eliminate the constant values c and a. In this respect, the normalized form of *nru* is as follows:

$$NRU = \sum_{j=1}^{N} nbu_{F_jC} \cdot \frac{1}{nb_{FC}} \tag{5.12}$$

Where $nb_{FC} = \sum_{j=1}^{N} nb_{F_jC}$.

Following from the above, we aim at minimizing the overall resource usage cost RU defined as the weighted sum of CRU and NRU:

$$\text{minimize} \qquad RU = w_c \cdot CRU + w_n \cdot NRU \tag{5.13}$$

$$\text{subject to} \qquad cmu_{F_j} \leq cm_{F_j}\ j = \{1, \ldots, N\}, \tag{5.14}$$

$$B \leq Bmax. \tag{5.15}$$

Where $w_c \geq 0$ and $w_n \geq 0$ are respectively the weights for the computational and network resource usage costs, which enable to specify a usage preference for one of the two types of resources over the other. In this work, we assume no such preference ($w_c = w_n = 1$).

Given the application graph G and the set of edge-cuts in G, we observe that we can select only one edge-cut ec_j as the replication and migration point per data stream S_j and Fog node F_j. We observe also that the resulting subgraph $Gmig_j$, delimited by ec_j, to replicate on the Fog node F_j for partially processing the data stream S_j there, requires the computational resource usage cmu_{Fj} and network bandwidth usage nbu_{F_jC} for sending the processed data stream S_j from the Fog node F_j to the Cloud node C. Thus, each individual selection of the replication and migration point ec_j per each data streams S_j and Fog nodes F_j produces different values of cmu_{Fj} and nbu_{F_jC}.

For each selected replication and migration points ec_j per each data streams S_j and Fog nodes F_j, the resulting cmu_{Fj} and nbu_{F_jC} enable to individually calculate the effect of this selection on the overall resource usage cost RU. Thus, we assume that the computational and network resource usage costs can be split for each individual data stream S_j as follows:

$$RU = \sum_{j=1}^{N} RU_j, \text{ where } RU_j = CRU_j + NRU_j \tag{5.16}$$

Where RU_j, CRU_j and NRU_j are respectively the contributions to RU, CRU, and NRU for processing a data stream S_j.

In this respect, to solve our SOO problem we need to search in the application graph G the replication and migration points ec_j for each individual data streams S_j and Fog nodes F_j such that RU should be minimized, each $Gmig_j$ should satisfy the computational resource usage constraint in (5.14) and the operator replicability constraint. On the other hand, the processed data streams S_j that will be transmitted on the Fog to Cloud network links should jointly satisfy the network bandwidth usage constraint (5.15).

Considering our SOO problem, the objective of minimizing RU, given the values of RU_j at each edge-cut ec_j and for all data streams S_j and Fog nodes F_j, under the global constraint of network bandwidth usage, the SOO problem can be mapped to a 0/1 knapsack problem [152], which is NP-hard. An exhaustive search approach is not tractable, particularly for a large search space, i.e., a big size of application graph G and/or a high number of Fog nodes and data streams. Hence, specialized algorithms that quickly rule out large parts of the search space or approximation algorithms should be used instead.

5.4.2 SOO-CPLEX Algorithm

We initially propose an approach that solves optimally our SOO problem by modelling it as an ILP problem. To this end, we introduce the decision variables X_{jk} to set an edge-cut $ec_k \in Grep$ as the replication and migration point of a data stream S_j that is to be processed on the Fog node F_j. X_{jk} is equal to 1 if ec_k is the selected edge-cut for the data stream S_j, otherwise X_{jk} is equal to 0. We consider the subgraph $Gmig_j \in Grep$ delimited by the edge-cut ec_k as the candidate subgraph to replicate and migrate on the Fog node F_j for processing the data stream S_j. Then, both computational and network resource usage costs defined respectively in (5.1) and (5.3) become:

$$cru = \sum_{j=1}^{N}\sum_{k=1}^{K} cmu_{F_{jk}} \cdot \frac{1}{cm_{F_j}} \cdot X_{jk} \tag{5.17}$$

$$nru = c + a \cdot \sum_{j=1}^{N}\sum_{k=1}^{K} nbu_{F_jC_k} \cdot \frac{1}{nb_{FC}} \cdot X_{jk} \tag{5.18}$$

Where N is the number of Fog nodes F_j and K is the number of edge-cuts identified in the replicable subgraph $Grep \subset G$. $cmu_{F_{jk}}$ is the computational resource usage for a candidate subgraph $Gmig_j$ delimited by the edge-cut ec_k on the Fog node F_j. $nbu_{F_jC_k}$ is the Fog-to-Cloud network bandwidth usage from the Fog node F_j to the Cloud node C when the edge-cut ec_k is considered as the migration and replication point in Grep.

Given the decision variable X_{jk}, and given the reformulation of cru and nru respectively in Formula (5.17) and Formula (5.18), we formulate the SOO problem as an ILP problem as follows:

$$\text{minimize} \qquad RU = wc \cdot CRU + wn \cdot NRU \tag{5.19}$$

$$\text{subject to} \qquad \sum_{k=1}^{K} cmu_{F_{jk}} \cdot X_{jk} \leq cm_{F_j} j = 1, \ldots, N, \tag{5.20}$$

$$\sum_{j=1}^{N}\sum_{k=1}^{K} nbu_{F_{jk}C} \cdot X_{jk} \leq Bmax \tag{5.21}$$

$$\sum_{k=1}^{K} X_{jk} = 1, j = 1, \ldots, N. \tag{5.22}$$

where $w_c = w_n = 1$. Formula (5.22) ensures that only one edge-cut ec_k is set as the replication and migration point in the subgraph Grep per data stream S_j.

Integer Linear Programming is known to be NP-hard [153]. To solve the SOO problem, we use the CPLEX solver; we call this approach SOO-CPLEX.

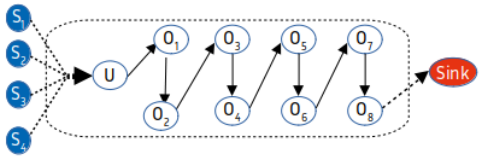


Figure 5.1: DSPA application used in the evaluation

Table 5.1: Parameters of the DSPA application of Figure 5.1

Operators (O_x):	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8
Selectivity (sel_x):	0.8	0.7	0.6	0.8	0.5	0.4	0.6	0.7
Cost (c_x):	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8

5.4.2.1 Experimental Evaluation

In this set of experiments, we compare SOO-CPLEX against the baseline RCS algorithm. In this respect, we use the simulator tool iFogSim [85]. iFogSim enables modeling both the hierarchical Edge-Fog-Cloud architecture and the DSPA application. It also enables implementing in Java different scheduling strategies. We implement the algorithms RCS and SOO-CPLEX.

To ensure that SOO-CPLEX produces the optimal solution, we set the optimality gap (OG) to 0.0%. OG is a metric used by CPLEX to trade off between optimality and performance.

Simulated Edge-fog-Cloud architecture and DSPA application We set in iFogSim an Edge-Fog-Cloud architecture comprising 4 Fog nodes, 1 Cloud node, and 4 IoT areas at the Edge. Each individual IoT area A_j produces an aggregated data stream S_j. Thus, the 4 IoT areas A_1, A_2, A_3 and A_4 produce the data streams S_1, S_2, S_3 and S_4 towards the closest Fog node F_1, F_2, F_3 and F_4, respectively. Each Fog node forwards further its data stream (that can be or not partially processed) to the Cloud node.

We set also in iFogSim the DSPA application as depicted in Figure 5.1. The cumulated operator selectivity is decreasing and the cumulated operator cost is increasing as we go from the source to the sink (see Table 5.1).

Dynamic IoT data stream rates To simulate, the random behaviour of the rates of the data streams S_1, S_2, S_3 and S_4, we create arbitrarily a set of 9 total data stream rate values in the interval [512, 2256] (in KB/sec). We wish to have around 15 results of the resource usage costs for each total data stream rate value and plot the average of these results per total data stream rate value. In this respect, we create a suite of 135 (9*15) instances of the 9 selected total data stream rate values that evolve randomly as depicted in Figure B.1. For each total data stream rate value instance, we set an interval [0, total data stream rate] in which we randomly

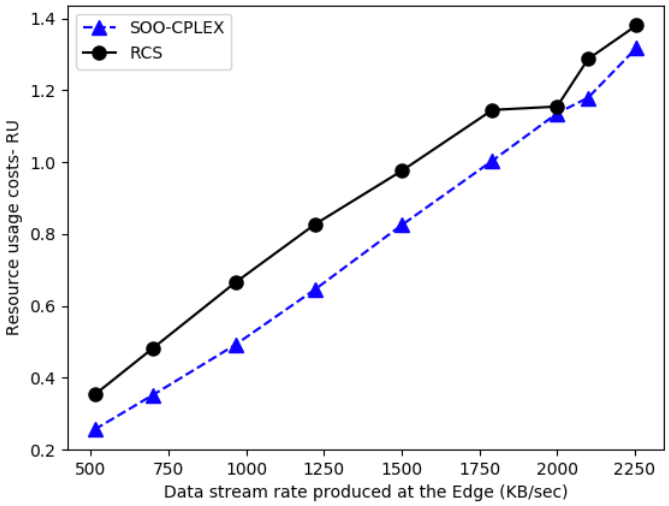


Figure 5.2: Overall resource usage cost

(uniform distribution) distribute the rate of each one of the data streams, so that the sum-rate is equal to the total data stream rate.

We feed the resulting suite of random data stream rates of the streams S_1, S_2, S_3, S_4 to RCS and SOO-CPLEX. For RCS, we maintain the deployment state of the DSPA application between the successive experiments, while for SOO-CPLEX each experiment is independent as SOO-CPLEX does not take into account this state.

Setting parameters For RCS, we set $Bmin = 800KB/sec$ and $Bmax = 1450KB/sec$ so that we have at least 2 total data stream rate values lower than $Bmin$ and 2 others between $Bmin$ and $Bmax$. Furthermore the evaluation is carried in two cases: (i) with costly (i.e., relatively more constrained) Fog computational resources in terms of memory, where we set $cm_{F_j} = 1280KB$ for each Fog node; and (ii) with less costly Fog resources, where we set $cm_{F_j} = 2048KB$ for each Fog node.

For the evaluation metrics, we compare the metrics RU, CRU and NRU of each RCS and SOO-CPLEX . In this respect, we consider as modest rate values, for the total data stream rate produced at the Edge, values that are lower than $Bmax$ (512KB/sec to 1221KB/sec), and as higher rate values, values that are higher than $Bmax$ (1500KB/sec to 2256KB/sec).

Evaluation results in the case of less costly Fog computational resources Figure 5.2 shows that whatever the data stream rates, SOO-CPLEX outperforms RCS, with a gap up to 37% at modest data stream rates and up to 18% at higher data stream rates.

Figures 5.4 and 5.3 show as expected that RCS favors using network resources until reaching Bmax and then it has to use also fog resources. SOO-CPLEX favors using Fog computational

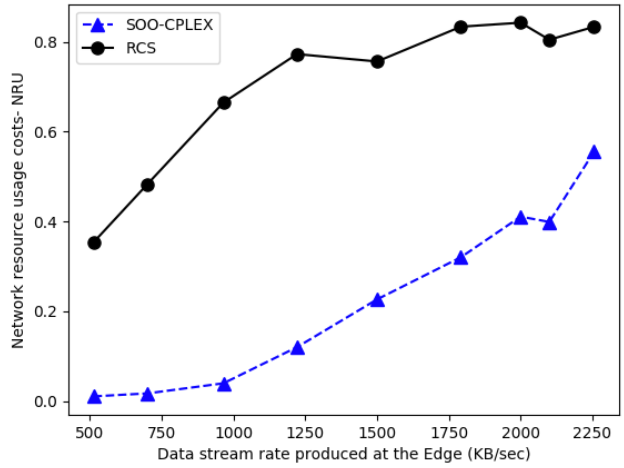


Figure 5.3: Network resource usage cost

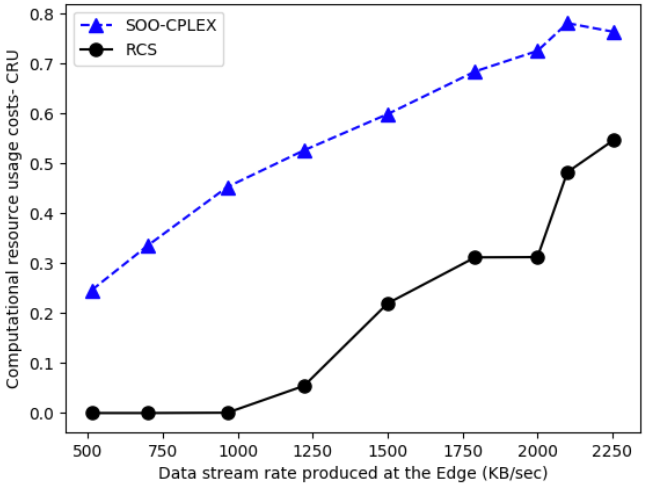


Figure 5.4: Computational resource usage cost

resources because apparently they are less costly than network resources.

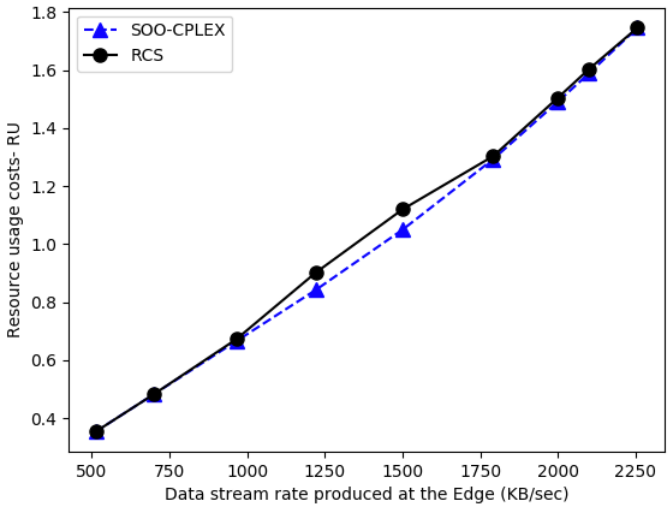


Figure 5.5: Overall resource usage cost

Evaluation results in the case of costly Fog computational resources At modest data stream rates, Figure 5.5 shows that RCS performs like SOO-CPLEX in terms of RU for data stream rates lower than $Bmin$. For modest data stream rates greater than $Bmin$, SOO-CPLEX has a slightly lower RU, hence outperforms RCS with a gap up to 0.6%. At higher data stream rates, SOO-CPLEX again slightly outperforms RCS by up to 0.6%. However as long as data stream rates increase, the difference between SOO-CPLEX and RCS is getting smaller.

Figures 5.7 and 5.6 show that SOO-CPLEX balances CRU and NRU, since at modest data stream rates it minimizes CRU so that its value is null and constant while NRU is monotonically increasing up to 0.84 as long as the modest data stream rates are lower than $Bmax$. However for higher data stream rates, the behaviour of SOO-CPLEX is reversed, CRU is increasing monotonically but in small proportions up to 0.74 while NRU has high values up to 0.9 but keep constant in the form of a tray, until the Fog computational resources are no longer sufficient to serve the maximum data stream rate 2256KB/sec, then $NRU > 1.0$.

5.4.3 SOO-H Algorithm

SOO-CPLEX provides optimal solutions to the scheduling problem we target. However, given the NP-hardness of the problem, it may incur a high execution time for large problem sizes. Hence, this solution is not appropriate for dynamic scheduling of operators in response to highly dynamic data streams. In this section, we introduce a heuristic algorithm, which we call SOO-H. SOO-H approximates the optimal operator scheduling with a lower execution cost. SOO-H is inspired from BOSe [111].

As presented in Algorithm 5, SOO-H first applies a parallel search to generate a solution that attempts to minimize the overall resource usage cost RU by minimizing independently RU_j

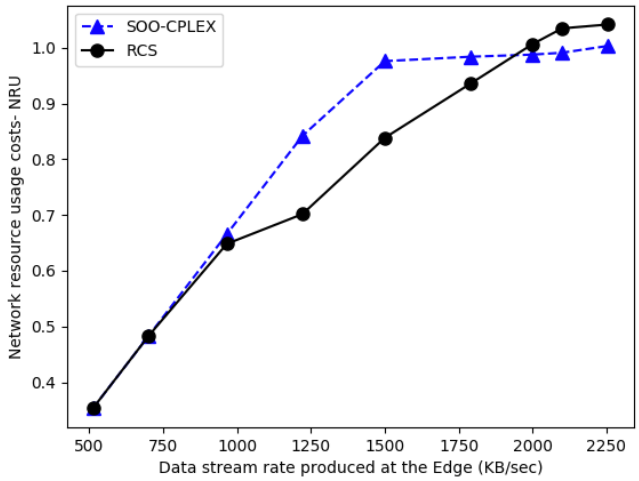


Figure 5.6: Network resource usage cost

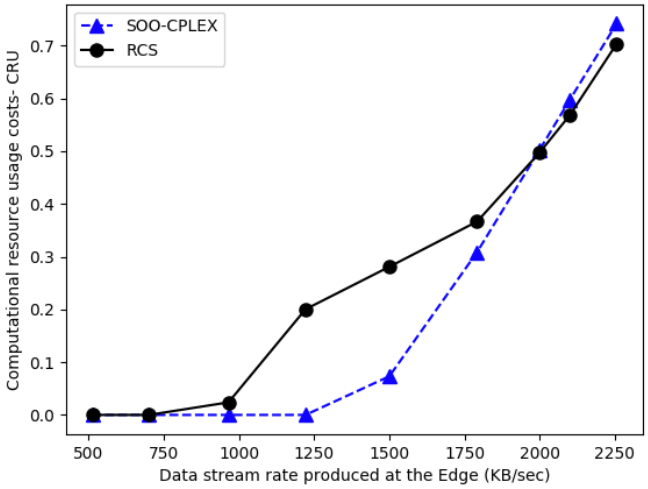


Figure 5.7: Computational resource usage cost

for each data stream S_j (line 6), with the function RUminCut(). If this solution satisfies the constraint $B \leq Bmax$, the achieved RU is optimal. Then, SOO-H rewrites G_{dep} to include all the $Gmig_j$ and deploys it across the Edge-Fog-Cloud (line 7, 23-24). Otherwise the problem may not have a solution or, if a solution exists, finding the optimal solution is NP-hard.

Thus in a second step, SOO-H applies a greedy search that produces local optimal solutions to approximate the global optimal solution in a reasonable amount of time. In this respect, we first use the minimum edge-cut approach to identify the solution that minimizes NRU by minimizing independently NRU_j for each data stream S_j (line 8), with the function dataMinCut(). If this solution does not satisfy the constraint $B \leq Bmax$, the problem has no solution. By relaxing accordingly the $Bmax$ constraint, we may accept this last solution as the best possible one. On the other hand, if the constraint $B \leq Bmax$ is satisfied, SOO-H applies a greedy search to

Algorithm 5: SOO-H

```
   Input: G, application graph
   Input: Grep, subgraph of replicable operators of G
   Input: Bmax, upper threshold for bandwidth usage
   Input: Sraw, set of raw data streams S_j
 1 Updated ← ∅, set of S_j on which ΔRU is applied S_j
 2 M ← ∅, set of Gmig_j, replicable subgraph per S_j
 3 RM ← ∅ set of edge-cuts ec_j ∈ Grep per S_j
 4 RU ← 0, overall resource usage cost
 5 B ← 0, Fog-to-Cloud network bandwidth usage
 6 M ← RUminCut()
 7 if B > Bmax then
 8     Selected ← dataMinCut()
 9     if B ≤ Bmax then
10         ΔRUset ← edgeCutMove(Selected, RM, M)
11         Sort ΔRUset in increasing order
12         Pull ΔRU on top of ΔRUset
13         while ΔRU < 0 do
14             Get S_j, e_{j_k}, Gmig_{j_k} corresponding to Δ_RU
15             ec_j ← MR[j]
16             if B − |ec_j| + |ec_{j_k}| ≤ Bmax and Updated.contains(S_j) == False then
17                 MR[j] ← e_{j_p}
18                 M[j] ← Gmig_{j_k}
19                 B ← B − |ec_j| + |ec_{j_k}|
20                 RU ← RU + Δ_RU
21                 Updated ← Updated ∪ S_j
22             Pull ΔRU on top of ΔRUset
23 Rewrite G_dep to include all Gmig_j (or Gmig_{j_p}) ∈ M
24 Deploy the new G_dep
```

improve this solution by reducing RU (lines 10-22).

Algorithm 6: SOO-H::RUminCut

```
1  Function RUminCut(Sraw, RM, M, RU, B, Grep):
2      for S_j ∈ Sraw do
3          Get Fog node F_j to serve S_j
4          Identify Gsat_j ⊆ Grep while cmu_{F_j} ≤ cm_{F_j}
5          Find ec_j in Gsat_j with the minimum RU_j ← CRU_j + NRU_j
6          Find Gmig_j ⊆ Gsat_j delimited by ec_j
7          M[j] ← Gmig_j
8          RM[j] ← ec_j
9          RU ← RU + RU_j
10         B ← B + |ec_j|
11         Keep and map ec_j to RU_j
12     return M
```

5.4.3.1 Parallel search

We use the function **RUminCut**; its pseudo code is presented in Algorithm 6. More specifically, for each data stream S_j we identify $Gsat_j \subseteq Grep$ as the part of $Grep$ that satisfies the computational resource constraint on the Fog node F_j receiving the data stream S_j. Then, we select the edge-cut $ec_j \in Gsat_j$ that produces the minimum RU_j (lines 3-5). We identify the candidate sub-graph $Gmig_j \subseteq Gsat_j$ to replicate on the Fog node F_j delimited by the identified edge-cut ec_j, and we set this edge-cut as the replication and migration point of S_j on the Fog node F_j (line 6-8). Finally, we update the Fog-to-Cloud network bandwidth usage B and the overall resource usage cost RU (line 9-11).

Algorithm 7: SOO-H::dataMinCut

Function `dataMinCut`(*Sraw, RM, M, RU, B, Grep*)**:**
  Selected $\leftarrow \emptyset$
  for $S_j \in Sraw$ **do**
    Get Fog node F_j *to serve* S_j
    Identify $Gsat_j \subseteq Grep$ *while* $cmu_{F_j} \leq cm_{F_j}$
    Find minimum ec_j *in* $Gsat_j$
    Find $Gmig_j \subseteq Gsat_j$ *delimited by* ec_j
    Compute cmu_{F_j} *and normalize to* CRU_j
    Compute nbu_{F_jC} *and normalize to* NRU_j
    $RU_j \leftarrow CRU_j + NRU_j$
    $RU \leftarrow RU + RU_j$
    $B \leftarrow B + |ec_j|$
    $M[j] \leftarrow Gmig_j$
    $RM[j] \leftarrow ec_j$
    Keep and map ec_j *to* RU_j
    Selected $\leftarrow$ *Selected* $\cup S_j$
  return *Selected*

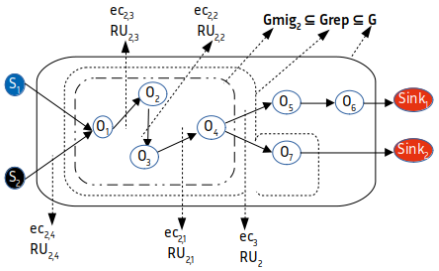


Figure 5.8: Example of edge-cut move

5.4.3.2 Greedy search

dataMinCut Similarly to *RUminCut*, for each data stream S_j, we identify $Grep \subseteq G$ and $Gsat_j \subseteq Grep$ for selecting, in this case, the minimum edge-cut ec_j that produces the minimum NRU_j. In this way, we identify the resulting candidate subgraph $Gmig_j \subseteq Gsat_j$ delimited by the minimum edge-cut ec_j. Then, we compute the overall NRU, RU and B. The pseudo code of *dataMinCut* is presented in Algorithm 7.

edgeCutMove Departing from the solution produced by *dataMinCut* and 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).

For example, in Figure 5.8 we identify $Gmig_2$ as the subgraph to replicate on the Fog at the minimum edge-cut ec_3 for processing the stream S_2. The possible backward edge-cut moves to consider are: ec_{2_1}, ec_{2_2}, ec_{2_3} and ec_{2_4}; the contributions to RU are respectively RU_{2_1}, RU_{2_2}, RU_{2_3}, RU_{2_4}.

Algorithm 8: SOO-H::edgeCutMove

```
1  Function edgeCutMove(Selected, RM, M, B):
2      ΔRUset ← ∅
3      for S_j ∈ Sraw do
4          ec_j ← RM[j]
5          Get RU_j corresponding to edge-cut ec_j
6          while ec_{j_k} is valid do
7              Gmig_j ← M[j]
8              Get Gmig_{j_k} ⊆ Gmig_j delimited by ec_{j_k}
9              Compute cmu_{F_j} and normalize to CRU_j
10             Compute nbu_{F_jC} and normalize to NRU_j
11             RU_{j_p} ← CRU_{j_k} + NRU_{j_k}
12             ΔRU ← RU_{j_k} − RU_j
13             ΔRUset ←ΔRUset ∪ΔRU
14             Keep and map ΔRU, S_j, Gmig_{j_k} and ec_{j_k}
15             ec_j ← ec_{j_k}
16             Get ec_{j_k} preceding ec_j ∈ Gmig_j
17     return ΔRUset
```

Given the above, for any backward edge-cut move ec_j on a data stream S_j, we assume the changes ΔCRU and ΔNRU respectively in CRU and NRU, where ΔCRU is in general negative and ΔNRU may be negative or positive. Thus, we can calculate the change in the resource usage cost RU as follows:

$$\Delta RU = \Delta CRU + \Delta NRU \tag{5.23}$$

The objective of the greedy search is first to identify all possible backward edge-cut moves, as produced by *edgeCutMove* (Algorithm 8), then to apply the edge-cut moves that produce the smallest $\Delta RU < 0$ (i.e., the ones that reduce RU the most), while satisfying the constraint $B \leq Bmax$.

To apply the edge-cut moves, in the SOO-H algorithm (Algorithm 5), we retrieve ΔRUset, the result of *edgeCutMove*, which is the set of ΔRU's, then we sort this set in increasing order, then we pull from the top of the set the smallest ΔRU (Algorithm 5, lines 10-12). If ΔRU is lower than 0, we set its corresponding edge-cut move ec_{j_k} as the current replication and migration point of the data stream S_j, and we set $Gmig_{j_k}$ as the current subgraph to deploy in the Fog, only if the constraint $B \leq Bmax$ is satisfied (Algorithm 5, lines 13-21). Then, we pull from ΔRUset the next lowest ΔRU to continue improving RU (Algorithm 5, line 22), as long as ΔRUset is not empty and we do not encounter a ΔRU ≥ 0. It is worth noting that we update at most once each data stream S_j with its best (lowest) ΔRU (Algorithm 5, line 16).

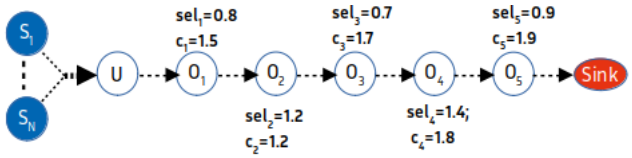


Figure 5.9: DSPA application used in the evaluation

5.5 Experimental Evaluation

We use the simulator tool iFogSim driven by realistic parameter settings that are proposed in [120]. As benchmarks, we consider: (i) SOO-CPLEX that produces optimal solutions, and (ii) RCS as baseline approach. In this respect, we implement SOO-H, SOO-CPLEX and RCS in Java inside the simulator. We set the CPLEX optimality gap to $OG = 0.0\%$ to ensure that SOO-CPLEX produces the optimal solution.

For our simulation experiments, we build a model of the TLC application (New York City Taxi and Limousine Commission rides) [154]). The TLC application finds the busiest taxi driver every two hours, where each vehicle emits at the end of a ride a data record containing driver identification, pick-up and drop-off times and locations. It comprises 5 operators and 1 sink as depicted in Figure 5.9.

We also build a simulation model of the Edge-Fog-Cloud architecture presented in Figure 1.3, where we scale the numbers of Fog nodes and IoT devices.

5.5.1 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) 10 values of M (number of IoT devices) in the interval $[5000, 50000]$, where each IoT device produces data at a rate of 4KB/s [29]. Then for each value of M, we set an interval $[0, M]$ in which we randomly (uniform distribution) set $m_j(t)$ IoT devices per geographical area A_j, so that the sum of $m_j(t)$ is equal to M. In this respect, each Fog node F_j receives data stream S_j at rate $|S_j| = m_j(t) \times 4KB/s$. As we wish to have around 15 results of RU, CRU, and NRU for each value of $M \in [5000, 50000]$ and to plot the average of these results per value of M, we repeat the splitting of each value of M among geographical areas 15 times. In this respect, the total data rate reaching the Fog follows a random (uniform distribution) sequence of the 10 values of M $\times 4KB/s$ repeated 15 times as depicted in Figure B.2. We feed this sequence to SOO-H, RCS and SOO-CPLEX. For RCS, we maintain the previous state of the DSPA application between the successive experiments. For SOO-H and SOO-CPLEX, each experiment is independent, since these two approaches calculate each time a new operator placement without taking into account the previous deployment state.

Table 5.2: Parameters of the DSPA application of Figure 5.1

Operators (O_x):	O_1	O_2	O_3	O_4	O_5
$csel_x$	0.8	0.96	0.672	0.9408	0.84672
$sum\_csel\_c_x$	1.5	2.78	4.412	5.6216	7.40912

5.5.2 Parameter setting

We evaluate the performance of SOO-H against RCS and SOO-CPLEX in both best-case and worst-case executions of the SOO algorithm. Best-case execution is related to the set of experiments where SOO-H applies only $RUminCut$. Worst-case execution is related to the set of experiments where, additionally, SOO-H applies a greedy search including $dataMinCut$ and $edgeCutMove$.

In this respect, we observed that SOO-H executes in the worst case only in a narrow subspace of parameter settings. Thus, we generate parameter settings inside and outside this subspace by proceeding as follows.

Assuming that the graph G of the DSPA application is a two-degree graph (as the one of Figure 5.9), we introduce the following two parameters: i) the cost of operator O_y for processing a unitary data load entering the graph, based on the cumulated selectivity of its upstream operator O_x: $csel\_c_y = csel_x \cdot c_y$, where c_y is the cost of the operator O_y; and ii) the cumulated sum of such costs up to an operator O_y: $sum\_csel\_c_y = sum\_csel\_c_x + csel\_c_y$.

Table 5.2 shows the values of $csel_x$ and $sum\_csel\_c_x$ for the DSPA application of Figure 5.9. Given the normalized forms of cru (i.e. CRU) and nru (i.e. NRU), we identify the following equilibrium, which represents a balance between the maximum CRU and NRU values over all possible edge-cuts selected as replication and migration points:

$$\frac{\underset{O_x \in G}{max}(sum\_csel\_c_x)}{\sum_{j=1}^{N} cm_{F_j}} \approx \frac{\underset{O_x \in G}{max}(csel_x)}{Bmax} \tag{5.24}$$

This equilibrium qualifies the case when, while moving from an edge-cut to another, network resources are replaced by computational resources of, more or less, equal cost. Then $RU = CRU + NRU$ will be almost constant.

We have observed that, for a best-case execution of SOO-H, the Fog computational resources should be less costly (i.e., relatively more abundant) than the Cloud network resources. To do so, we increase cm_{F_j} for each Fog node F_j or decrease $Bmax$ with respect to the equilibrium (5.24). This ensures that $RUminCut$ produces a solution where the constraint $B \leq Bmax$ is satisfied.

Accordingly, we consider N=30 Fog nodes, and we set their computational capacity in terms of RAM so that 20% of the nodes have 256MB, 30% have 1GB and 50% have 2GB. Then we set

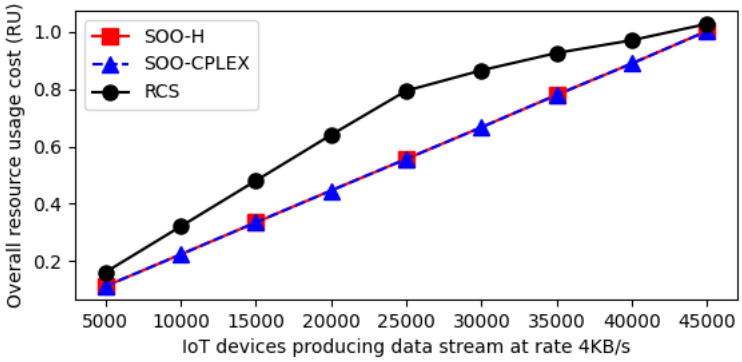


Figure 5.10: Overall resource cost (best-case execution)

$Bmax = 125000KB/s$ ($\approx 1Gbps$).

For a worst-case execution of SOO-H, the Fog computational resources should be more costly (i.e., relatively less abundant) than the Cloud network resources. To this end, we decrease cm_{F_j} for each Fog node F_j or increase $Bmax$ with respect to the equilibrium (5.24). However, we cannot go very far away from the equilibrium, otherwise by decreasing too much cm_{F_j} of each Fog nodes F_j or increasing too much $Bmax$, $dataMinCut$ may produce a solution with $B > Bmax$, and hence the TSOO algorithm will produce a solution where the constraint $B \leq Bmax$ is not satisfied. Accordingly, we consider N=10 Fog nodes where all of them have 256MB of RAM, and we keep $Bmax = 125000KB/s$. Additionally for the baseline solution RCS, we set $Bmin = 75000KB/s$.

5.5.3 Evaluation results

We distinguish the evaluation results between the best-case and the worst-case executions. In the former, SOO-H applies RUminCut, which directly identifies the optimal solution. In the latter case, SOO-H in contrary applies greedy search with dataMinCut and edgeCutMove, in which an approximation of the optimal solution is identified.

5.5.3.1 Best-case execution

In this set of experiments, the Fog computational resources are relatively abundant and hence their usage is less costly than the usage of the Cloud network resources. As depicted in Figure 5.10, whatever the number of IoT devices at the Edge and consequently the data stream rates reaching the Fog nodes, SOO-H performs like SOO-CPLEX, thus it finds the optimal RU with $RUminCut$, for which the resulting solution satisfies directly the constraint $B \leq Bmax$. When comparing to RCS, SOO-H outperforms it with a difference ratio that lies between 43.96% and 2.48% with respect to the evolution of the number of IoT devices at the Edge.

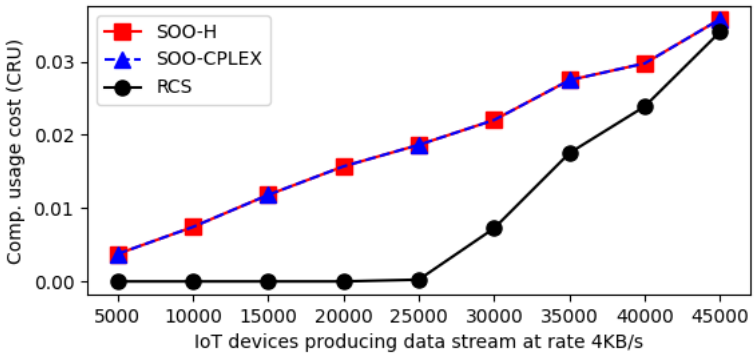


Figure 5.11: Computational resource cost (best-case execution)

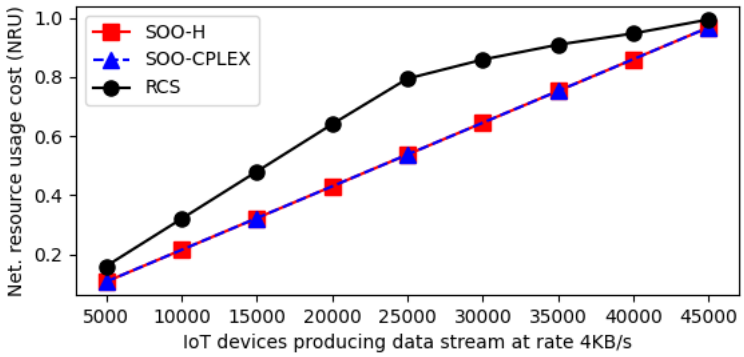


Figure 5.12: Network resource cost (best-case execution)

Figure 5.11 and Figure 5.12 show that SOO-H (same as SOO-CPLEX) finds a balance between CRU and NRU in order to minimize RU. On the other hand, RCS resorts minimally to the Fog resources. When $M \leq 20000$ IoT devices, the resulting data stream rate that reaches the Cloud through the Fog nodes is lower than $Bmax$, hence no processing is moved to the Fog and CRU is null. At the same time, the counterpart in terms of NRU is high. When $M > 20000$ IoT devices, CRU for RCS increases for satisfying the constraint $B \leq Bmax$.

Furthermore, we can see in Figure 5.11 that CRU values of SOO-H, SOO-CPLEX and RCS are very small, lower than 0.1. This is due to the specific computational requirements of the DSPA application (Figure 5.9) and the high capacities of the Fog computational resources.

5.5.3.2 Worst-case execution

In this set of experiments, the Fog computational resources are pretty limited, thus their usage is more costly than the usage of the Cloud bandwidth.

As shown in Figure 5.13, the RU plots are steeper (compared to Figure 5.10) for RCS, SOO-H and SOO-CPLEX, with small differences between them. More specifically, SOO-H achieves a lower RU compared to RCS with a difference ratio ranging between 14.77% and 0.76% with

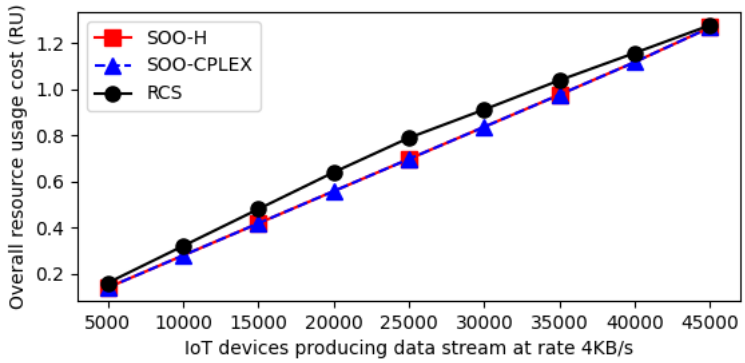


Figure 5.13: Overall resource cost (worst-case execution)

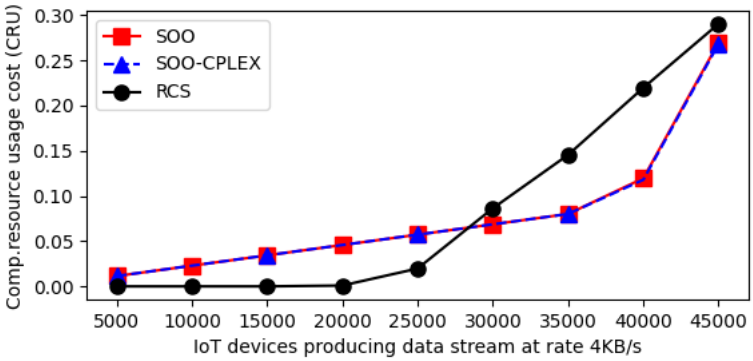


Figure 5.14: Computational resource cost (worst-case execution)

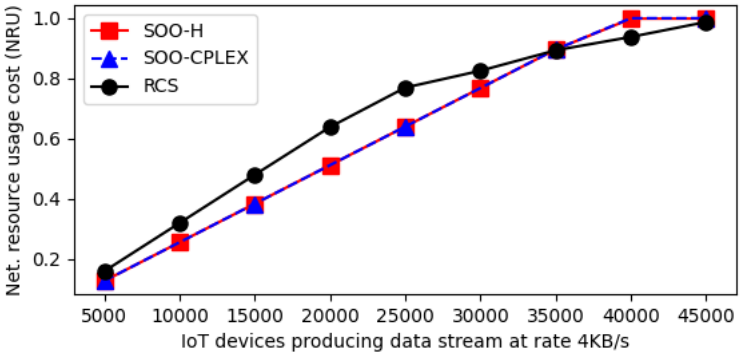


Figure 5.15: Network resource cost (worst-case execution)

respect to the number of IoT devices (M) at the Edge.

Furthermore, SOO-H achieves equal *RU* compared to SOO-CPLEX for the experiments where $M < 40000$ IoT devices at the Edge. This still corresponds to best-case execution for SOO-H, thus SOO-H produces optimal results like SOO-CPLEX. On the other hand, when M is equal to 40000 and 45000 IoT devices, SOO-CPLEX outperforms SOO-H, with a small

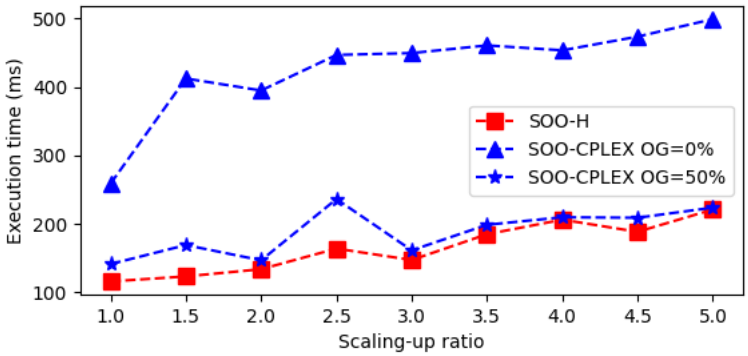


Figure 5.16: Execution time by scaling N, Bmax and M

difference ratio up to 0.01%. This corresponds to worst-case execution for SOO-H, where SOO-H approximates the optimal solution.

Figure 5.14 and Figure 5.15 show a behavior for RCS in terms of CRU and NRU that is similar to Figures 5.11 and 5.12. On the other hand, SOO-H and SOO-CPLEX achieve a smoother and more balanced use of the two types of resources, even with limited Fog resources.

5.5.3.3 Scalability analysis

We assess the scalabilty of SOO-H against SOO-CPLEX in terms of execution time, i.e., the time it takes for either algorithm to find the best operator placement. In this respect, we use as starting point the parameter settings for SOO-H worst-case execution with $N = 10$ Fog nodes, $M = 40000$ IoT devices and $Bmax = 125000KB/s$. To maintain the same ratio in parameter setting with respect to the equilibrium (5.24), we consider the scaling up ratio $\frac{N}{N_{init}}$, where $N_{init} = 10$ Fog nodes, then we scale $Bmax$ and M by multiplying their initial values with the scaling up ratio for each increase of N, where initially $N = N_{init}$.

Besides the setting of SOO-CPLEX with $OG = 0.0\%$ (see Section 5.4.2.1), we additionally consider the setting of SOO-CPLEX with $OG = 50\%$ to enable the CPLEX tool to balance between optimality and execution time.

As depicted in Figure 5.16, for the different scaling up ratios, SOO-H has lower execution times, with a difference ratio ranging between 54% and 70%, compared to SOO-CPLEX with $OG = 0\%$. Even when compared to SOO-CPLEX with $OG = 50\%$, SOO-H has lower execution times with a difference ratio between 1% and 30%.

In terms of the produced RU solution, Figure 5.17 shows that SOO-H fails to find the optimal solution compared to SOO-CPLEX with $OG = 0\%$, but the approximation error is very small, ranging from 0.01% to 0.0003%, following the scaling-up ratio increase. On the other hand, when comparing to SOO-CPLEX with $OG = 50\%$, SOO-H produces a better RU solution whatever the scaling-up ratio.

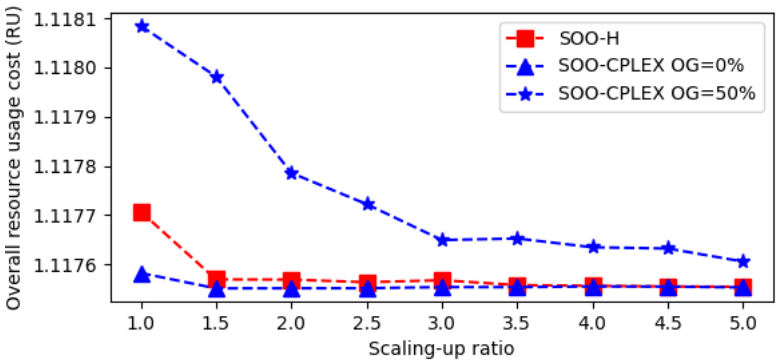


Figure 5.17: Resource usage cost by scaling N, Bmax and M

5.6 Conclusion

In this chapter, we designed and implemented resource aware scheduling algorithms that statically deploys DSPA application between the Fog and Cloud nodes in order to process the dynamic data stream rates produced by IoT devices at the Edge. These algorithms rely on our holistic resource usage cost model that uses the static weight version.

In this respect, the baseline RCS algorithm does not optimize the resource usage cost, it only satisfies the problem constraint by using as less as possible the Fog computational resources. However, by solving our problem as a single objective optimization problem, the main conclusions drawn from our experiments follows. First, compared to simple baseline such as RCS, SOO-CPLEX and SOO-H are not only capable in finding scheduling solutions that satisfy the resource usage constraints, but also that exhibit interesting trade-offs between the usage of the Fog computational resources and the usage of Cloud network bandwidth resources so that the sum of the two costs is minimized.

Second, SOO-H identifies optimal solutions like SOO-CPLEX when Fog computational resources are abundant. Even though that for very limited (or constrained) computational resources at the Fog layer, SOO-H may fail to approximate the optimal solutions found by SOO-CPLEX. However, the approximation error is very small up to 0.01%.

In overall, we observe that the resource usage cost model achieve enough to respond to a resource-aware QoS requirement for the static deployment of DSPA application at the IoT network edge featuring constrained and heterogeneous computational resources. However, due to these constrains, replicating operators at Fog nodes may increase their processing times even if the reduced data streams sent toward the Cloud will lower the involved network delays. This observation fosters us to consider not only the resource usage cost model but also the response time model of DSPA applications distributed between the Fog and the Cloud. For these reasons, in the next Chapter 6, we will extend the formulation of our optimization problem to take into

account the response time constraints of DSPA applications.

Chapter 6

Resource-Aware Scheduling of Continuous Operators With Time Constraints

Contents

This chapter contains 19 pages.

6.1 Introduction

In this chapter, we extend the SOO problem introduced in Chapter 5 with response time constraints of DSPA application, and hence we formulate the extension of SOO problem as a Time based Single Objective Optimization (TSOO) problem. To take into account the response time of DSPA application, we rely on the response time model presented in Chapter 4 (see Section 4.4).

The remainder of this Chapter is organized as follows: In Section 6.2 we formulate the TSOO problem and show that we can map this problem to a job shop scheduling (JSS) problem known to be NP-hard [155]. In Section 6.3, we first solve the TSOO problem by using meta-heuristic algorithm based on simulated annealing (SA) (TSOO-SA) [143]. Next, in Section 6.4 we introduce a heuristic algorithm called TSOO-H which extends our SOO-H algorithm in order to take into account the response time constraint. In Section 6.5 we experimentally evaluate TSOO-SA and TSOO-H. Finally, Section 6.6 concludes this chapter.

The contributions described in this Chapter have been published in the International Conference on Smart Computing (Smartcomp2022) [156].

6.2 Time based Single Objective Optimization (TSOO)

Prior to formulate the TSOO problem, we need to adjust the response time model and the network resource usage model.

6.2.1 Adjusting the models

As TSOO problem extends the SOO problem that assumes the data streams are processed only between the Fog and Cloud, we need to formulate the response time model to cope with this assumption. Furthermore by introducing the response time model we need also to take into account the network delay of each individual Fog to Cloud network link that may have an impact on the network resource usage cost defined in Formula (5.3).

6.2.1.1 Response time model

We defined the response time T as the worst end-to-end latency $L\pi_{ij}$ among all the operator paths π_{ij} of the deployable application graph G_{dep}. In this Chapter we assume that the data streams are not processed at the Edge and the G_{dep} is distributed only between the Fog and Cloud nodes. In this respect the operator latency at the Edge is zero and hence the end-to-end operator path latency defined in Formula (4.19) becomes:

$$L\pi_{ij} = \max_{\forall E_i \in A_j(t)} (nd_{EiFj}) + \sum_{e_{xy} \in \pi_{ij}} nd_{\mathcal{M}(x),\mathcal{M}(y)} + \sum_{O_x \in \pi_{ij}} l_x \tag{6.1}$$

Where the first term 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.

Given that data streams are processed only between the Fog and Cloud, $\mathcal{M}$ is the mapping function, that gives the resource node (i.e. Fog node F_j or Cloud node C) on which 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 Fog node F_j that hosts the operator O_x to the Cloud node C that

hosts the operator O_y or the sink y. $nd_{\mathcal{M}(x),\mathcal{M}(y)}$ is negligible if the operator O_x and the operator O_y or the sink y are placed on the same resource node (i.e., Fog node F_j or Cloud node C). Otherwise it is not negligible. Furthermore, l_x is the latency of the operator O_x to process its input data load.

6.2.1.2 Resource usage cost

Unlike in SOO problem where the Fog to Cloud network delays are assumed to be equal (statically calculated). By introducing the response time, we need to take into account the network delay of transmitting data streams from the Fog to Cloud. This network delay along with the network bandwidth can be dynamic and hence, we need to take into account these two factors in order to differentiate these Fog to Cloud network links. Thus, in this Chapter we consider the overall network resource usage cost (nru) defined in Formula (5.2). However, in order to have nru without unit, we divide the network delay of each individual Fog to Cloud network links by the maximum network delay among all the network links. In this respect the overall network resource usage cost becomes:

$$nru = c + \sum_{j=1}^{N} nbu_{F_jC} \cdot \frac{1}{nb_{F_jC}} \cdot \frac{nd_{F_jC}}{nd_{max}} \tag{6.2}$$

Where $nd_{max} = \max_{i,j}\{nd_{E_iF_j}, nd_{F_jC}\}$ is the maximum network delay.

In order to normalize nru to NRU, we can eliminate the constant value c as it can not change during the optimization. In this respect, the cost of using each Fog to Cloud network link takes values between 0 and 1. Consequently nru takes values between 0 and N, as we have N ($N \geq 1$) Fog to Cloud network links across the Edge-Fog-Cloud architecture. In this respect the normalized form of nru is as follows:

$$NRU = \frac{nru}{N} \equiv \frac{\sum_{j=1}^{N} nbu_{F_jC} \cdot \frac{1}{nb_{F_jC}} \cdot \frac{nd_{F_jC}}{nd_{max}}}{N} \tag{6.3}$$

In this respect, NRU takes values between 0 and 1.

However, introducing the response time constraint in the problem does not require to adjust the formulation of the overall computational resource usage cost (i.e. cru). We consider cru defined in Formula (5.1) along with its normalized form CRU defined in Formula (5.11).

6.2.2 Problem formulation

While the scheduling of G as G_{dep} should take into account any response time constraint imposed by a DSPA application, we formulate the TSOO problem as following:

$$\text{minimize} \quad RU = w_c \cdot CRU + w_n \cdot NRU \tag{6.4}$$

$$\text{subject to:} \quad cmu_{F_j} \leq cm_{F_j}\ j = \{1, \ldots, N\}, \tag{6.5}$$

$$B \leq Bmax \tag{6.6}$$

$$T \leq Tmax. \tag{6.7}$$

To ensure the real time constraint, Formula (6.7) imposes that the response time of a specific DSPA application should not exceed a threshold $Tmax$.

6.2.2.1 Problem complexity

Our TSOO problem can be mapped to the job shop scheduling (JSS) problem. In JSS problem we have n jobs that will need to be processed using m machines. Each job consists of X tasks, each of which needs to be processed in a certain order. The objective is to find the scheduling of these n jobs on the m machines in order to ensure resource usage fairness, to minimize the completion time of the jobs or to ensure deadline constraint (i.e. completion time within a certain threshold).

To map the TSOO problem as an instance of JSS problem, we can consider the Fog and Cloud nodes as the m machines, each operator path $\pi_{ij} \in \mathrm{G}_{dep}$ as a job, each operator in the path as a task job and the edges between operators determine the operator processing order. Each operator requires computational and network resources to process and transmit data stream with a certain latency and network delay. Furthermore, we can consider the DSPA application response time as the completion time.

Finding the scheduling of operators between the Fog and Cloud nodes with the objective to ensure the response time constraint and fairness in resource usage [155], where the latter is defined in terms of: (i) optimal trade-off between CRU and NRU (cfr Formula (6.4)), (ii) resource usage constraints (cfr Formulas (6.5) and (6.6)) and (iii) operator replicability constraint. In this respect, our TSOO problem can be set as an instance of JSS problem with deadline constraint [155, 157]. The JSS problem is known to be NP-hard and hence, the TSOO problem is NP-hard as well. Its complexity increases as we increase the number of resource nodes, data streams and the size of the application graph G.

Solving the JSS problem consistently is still challenging [158]. In this respect, meta-heuristic based approaches are largely used as ways of obtaining better solutions, because of their flexibility and global optimization capabilities [158]. Meta-heuristics that have been used to solve the JSS problem include but are not limited to: (i) Genetic Algorithm (GA), used for example by Souleiman et al. [159] to solve the problem of optimal scheduling and allocation of jobs under a QoS requirement in terms of job waiting time on a multi-cloud tier; (ii) Simulated Annealing (SA), applied for example by Fang et al. [160] to solve the application modules placement problem between the Edge and Cloud while minimizing IoT application response time and energy

consumption; (iii) Tabu search (TS), used by Zang et al. [161] in combination with SA to solve time-efficiently the JSS problem; (iv) Particle swarm optimization (PSO), used for instance by Djemai et al. [162] to efficiently place IoT application modules on the Fog with the goal of minimizing energy consumption and application response time violations, subject to resource nodes capacity constraints and module placement constraints.

In this thesis, we do not seek to demonstrate the superiority of one meta-heuristic over the other. Even though each one of these approaches comes with its own limitations [158]. The common limitations of these approaches are parameter setting and high execution time. Therefore, we propose a greedy approach, TSOO-H, for solving the TSOO problem time-efficiently. To evaluate the quality of the results of TSOO-H, we also develop TSOO-SA, a solution based on the SA meta-heuristic to solve the TSOO problem. The choice of SA is motivated by the fact that it is designed to escape from the local minimum compared to TS thanks to its capability of accepting non local minimum solution. Moreover it requires less parameter setting compared to GA [161].

6.3 TSOO-SA algorithm

Simulated annealing (SA) is an iterative search method for approximating the global optimum for a given optimization problem [163]. Starting from the current solution with cost f_i and based on an adequate perturbation method to generate a neighbour solution of the current one with cost f_j, SA accepts the neighbourhood solution as the current one on the basis of a probability controlled by a parameter called temperature τ:

$$\mathcal{P}[\text{accept } f_j] = \begin{cases} 1, & \text{if } (f_j - f_i) < 0. \\ e^{-\frac{f_j - f_i}{\tau}}, & \text{otherwise.} \end{cases} \tag{6.8}$$

Besides the initial temperature τ at which SA starts, SA includes also [143]: (i) the cooling rate α, which determines how quickly the temperature decreases: $\tau = \tau \cdot \alpha$, where typically $\alpha \in [0.8; 0.99]$; (ii) a finite number of transitions for each value of τ, which determines after how many neighbour solutions generated SA will decrease the temperature; and (iii) the final temperature value τ_{final}, which determines the temperature at which SA will end.

To solve the TSOO problem using SA, we need to define the corresponding problem:

Search space any edge-cut $ec_j \in G$ that can be set as the replication and migration point of an individual data stream S_j to be processed on the Fog node F_j.

Solution the graph G_{dep} to be deployed across H and the corresponding operator placement $\mathcal{M}$ for an RM. RM is the set of selected edge-cuts ec_j to be each the replication and migration

point of a data stream S_j.

Cost function In the TSOO problem, we consider RU as the cost function to minimize, under the constraints of the Fog to Cloud bandwidth usage, the Fog computational resource usage, and the response time T calculated as the maximum end-to-end latency among all the operator paths $\pi_{ij} \in G_{dep}$.

It has to be noted that TSOO is a minimization problem with constraints, while SA is designed to solve unconstrained optimization problems. For this reason, we need to consider an approximation of the TSOO problem for SA. In particular, we consider a constraint relaxation method frequently used in this context [143]. We add to the cost function to minimize (i.e., RU) a penalty ζ_k for the violation of each individual constraint of the TSOO problem:

$$f = w_c \cdot CRU + w_n \cdot NRU + \sum_{k=1}^{k_{max}} \zeta_k \tag{6.9}$$

We set $\zeta_k = 1$ if $val_k - max_k > 0$, otherwise $\zeta_k = 0$. Where val_k can be the value of cmu_{Fj}, B or T and max_k can be the value of cm_{F_j}, $Bmax$ or T_{max}, respectively.

6.3.1 Algorithm description

The pseudo-code of the TSOO-SA algorithm is presented in Algorithm 9. We set $Grep$ as the subgraph of G containing the set of operators that can be replicated on the Fog. Given that SA starts from an initial solution which is likely to lead the search to converge towards the global optimum, TSOO-SA produces this solution by applying the function $dataMinCut()$ (presented in Algorithm 7), which identifies for each data steam S_j and Fog node F_j, the minimum edge-cut ec_j as the replication and migration point. This solution comprises: RM, the set of replication and migration points; and M, the set of subgraphs $Gmig_{n_i}$ to replicate and deploy on each individual resource node $n_i = F_j|C$. This solution becomes the current best solution with cost f (Algorithm 9, line 4). It also becomes the current solution to improve further, if it satisfies the constraint $B \leq Bmax$ (Algorithm 9, line 5).

For each set of L iterations, TSOO-SA applies the temperature value τ, while it decreases it progressively at the end of each set by using the cooling rate α (Algorithm 9, line 19). TSOO-SA stops when τ reaches the predefined lower temperature threshold, τ_{final}.

At each iteration (Algorithm 9, lines 6-19), a new solution is taken from the neighbourhood of the current solution. To generate a neighbour solution, we use the function neighboringSolution() (Algorithm 9, line 8), which is detailed in the next section. The cost of the new generated solution, f_l, is compared with that of the current solution, f_{curr}, in order to determine if an improvement has been achieved. If f_l is smaller than f_{curr}, the new solution becomes the current solution (Algorithm 9, lines 9-12). Furthermore, if f_l is smaller than the cost of the best solution,

Algorithm 9: TSOO-SA

```
   Input: G, application graph
   Input: Grep, subgraph of replicable operators of G
   Input: τ, initial temperature value
   Input: τ_final, final temperature value
   Input: α, cooling rate
   Input: L, maximum iteration number
   Input: Sraw, set of raw data streams S_j
 1 M ← ∅, set of Gmig_{n_i}, to replicate on per node n_i
 2 RM ← ∅ set of edge-cuts ec_j ∈ Grep per S_j
 3 f_cur, M_cur ← dataMinCut(Grep, Sraw)
 4 f ← f_cur  &  M ← M_cur
 5 if B ≤ Bmax then
 6     while τ > τ_final do
 7         for l = 1 to L do
 8             f_l, M_l ← neighboringSolution(Grep, RM)
 9             Δ ← f_l − f_cur
10             if Δ < 0 then
11                 f_cur ← f_l
12                 M_cur ← M_l
13                 if f_cur < f then
14                     f ← f_cur
15                     M ← M_cur
16             else if rand(0,1) ≤ P[accept f_l] then
17                 f_cur ← f_l
18                 M_cur ← M_l
19         τ ← τ · α
20 Rewrite G_dep to include all Gmig_ni ∈ M
21 Send each Gmig_ni ∈ M to the corresponding node n_i
```

f, the new solution becomes the best solution (Algorithm 9, line 13-15). On the other hand, if f_l is not smaller than f_{curr}, the new solution can be accepted as current solution (Algorithm 9, lines 16-18) with the probability defined in Formula (6.8). The probability for accepting a neighbour solution decreases as τ decreases. For high values of τ, the search is almost random, while for low τ values, the search becomes almost greedy.

6.3.2 Generating neighbour solution

The function *neighboringSolution*() is illustrated in Algorithm 10 in which we represent RM the set of replication and migration points as an array. At each array index of RM, we have the mapping data stream S_j and edge-cut ec_j to be selected as the replication and migration point of S_j. In order to avoid TSOO-SA to be trapped in a local minimum during the search [161], we guide the search so that the selection of the replication and migration points goes continuously from the source to the sink in G. Opposite to a random selection of a neighborhood solution (as in the mutation technique called shift move (SM) [164]) we only randomly select a data stream

Algorithm 10: neighboringSolution

```
1  Function neighboringSolution(Grep, RM, Sraw):
       Input: Grep, subgraph of replicable operators of G
       Input: Sraw, set of raw data streams S_j
       Input: RM, set of tuple <data stream, edge cut>
2      Random select S_j from Sraw
3      Get < S_j, ec_j > from RM that corresponds to S_j
4      index ← 1
5      while index < sizeOf(RM) do
6          < S_k, ec_k > ← RM[index]
7          if S_k == S_j then
8              Get ec'_j directly succeeding ec_j in Grep
9              RM[index] ←<S_j, ec'_j >
10         else if index == 1 then
11             RM[index] = RM[sizeOf(RM)]
12         else
13             RM[index+1] ←<S_k, ec_k >
14     return RM
```

S_j (Algorithm 10, line 2). From the selected data stream S_j, we select its current replication and migration point ec_j that we replace by ec'_j (Algorithm 10, lines 7-9). ec'_j is the edge-cut that directly succeeds ec_j as we go from the source to the sink of G. If ec_j is the last edge-cut in G (or Grep, the sub-graph of replicable operators), ec'_j will take the value of the first edge-cut in G (or Grep). Then, we move for all the data streams $S_k \neq S_j$, the replication and migration points from one position (i.e. array index) in RM. This move can be right or left, in the present work we consider right move (Algorithm 10, line 10-13).

Example of generating a neighbourhood solution: To illustrate the generation of a new solution in the neighbourhood of the current solution, let consider an application graph G (which is equal to its Grep). This G has the following edge-cuts: $ec_1, ec_2, ec_3, ec_4, ec_5$ and it process the data streams Sraw=$\{S_1, S_2, S_3\}$. If we consider that the function $dataMinCut()$ identifies the following initial solution that we represent as an array of replication and migration points: RM=$\{(S_1, ec_2); (S_2, ec_4); (S_3, ec_1)\}$.

To generate the neighbourhood solution by applying the Algorithm 10, we process as follows: randomly select S_2, as ec_5 succeeds directly ec_4 in the set of edge-cuts of G, we set ec_5 as the new replication and migration point of S_2 that replaces ec_4. Then, we move (to right) the replication and migration point of S_1 and S_3. This means that S_1 will have ec_1 as its current replication and migration point the one that S_3 had before and S_2 is not concerned as its has a new replication and migration point ec_5. Then S_3 will have ec_4 the previous replication and migration point of S_2. Thus, we have the following neighbour solution RM_1=$\{(S_1, ec_1); (S_2, ec_5); (S_3, ec_4)\}$.

6.4 TSOO-H Algorithm

To overcome the high execution cost of TSOO-SA, we propose a heuristic approach called *TSOO-H*. TSOO-H extends the SOO-H algorithm introduced in Chapter 5. It reuses the functions of SOO-H: RUminCut (cf. Algorithm 6), DataMinCut (cf. Algorithm 7), EdgeCutMove (cf. Algorithm 8).

6.4.1 Algorithm description

TSOO-H (cf. Algorithm 11) starts by applying $RUminCut()$ to generate a solution that attempts to minimize directly RU. If the output solution of $RUminCut()$ satisfies the constraints $B \leq Bmax$ and $T \leq Tmax$, then the achieved RU is optimal (Algorithm 11, line 6). Otherwise the problem may not have a solution or, if a solution exists, finding the optimal solution is NP-hard.

As a next step, TSOO-H applies a greedy search that produces local optimal solutions to approximate the global optimal solution in a reasonable amount of time. In this respect, we apply the function $dataMinCut()$ to identify the solution that minimizes NRU and consequently B (Algorithm 11, lines 7-8). If the output solution of $dataMinCut()$ does not satisfy the constraint $B \leq Bmax$, the TSOO problem has no solution satisfying this constraint, unless we relax $Bmax$ in order to accept this solution as the least bad one.

If the constraint $B \leq Bmax$ is satisfied, TSOO-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 subgraphs $Gmig_j$ and $Gmig_C$ the operators to move from the Fog to the Cloud (or the inverse) in order to reduce T until we satisfy this constraint, while keeping satisfied the constraint $B \leq Bmax$ (Algorithm 11, lines 9-13). This greedy search is performed with the functions $operatorMoveback()$ and $operatorMoveDown()$, which we describe in Section 6.4.1.1. 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 that 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 minimize RU while keeping satisfied the problem constraints (Algorithm 11, lines 14-30). We describe this greedy search in Section 6.4.1.2.

6.4.1.1 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

Algorithm 11: TSOO-H

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

$X \leftarrow \emptyset$, set of S_j on which ΔRU is applied S_j
$M \leftarrow \emptyset$, set of $Gmig_j$, replicable subgraph per S_j
$RM \leftarrow \emptyset$ set of edge-cuts $ec_j \in Grep$ per S_j
$RU \leftarrow 0$, overall resource usage cost
$B \leftarrow 0$, Fog-to-Cloud network bandwidth usage
$M \leftarrow RUminCut()$
if $B > Bmax$ **then**
    $M, RM, G_{dep} \leftarrow dataMinCut(G)$
if $T > Tmax \wedge B \leq Bmax$ **then**
    Set $max\pi$, sorted set of paths π_{ij} where $L\pi_{ij} > Tmax$
    $operatorMoveBack(M, RM, G_{dep}, max\pi)$
    if $T > Tmax$ **then**
        $operatorMoveDown(M, RM, G_{dep}, max\pi)$
if $B \leq Bmax \wedge T \leq Tmax$ **then**
    ΔRUset $\leftarrow edgeCutMove(G, RM, M)$
    Sort ΔRUset in increasing order
    Pull ΔRU on top of ΔRUset
    while $\Delta RU < 0$ **do**
        Get $S_j, e_{j_k}, Gmig_{j_k}$ corresponding to Δ_{RU}
        $ec_j \leftarrow RM[j]$
        Calculate T when considering ec_j
        $B^{'} \leftarrow B - |ec_j| + |ec_{j_k}|$
        if $B^{'} \leq Bmax \wedge T \leq Tmax \wedge X.has(S_j) = False$ **then**
            $RM[j] \leftarrow e_{j_p}$
            $M[j] \leftarrow Gmig_{j_k}$
            $B \leftarrow B - |ec_j| + |ec_{j_k}|$
            $RU \leftarrow RU + \Delta_{RU}$
            $X.add(S_j)$
        Pull ΔRU on top of ΔRUset
Rewrite G_{dep} to include all $Gmig_{nj}$ (or $Gmig_{j_p}$) $\in$ M
Send each $Gmig_{nj} \in M$ to corresponding resource node n_j

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 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 were 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.

Algorithm 12: 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
```

Operator move back In this respect, to satisfy the response time constraint we first apply the function $operatorMoveback()$, as it is more likely to reduce the response time T on an operator path.

More specifically as presented in Algorithm 12, we iteratively select the operator paths π_{ij} where $L\pi_{ij} > Tmax$, in decreasing order of their end-to-end latencies $L\pi_{ij}$. For each selected π_{ij}, we start from the edge-cut ec_j that delimits the two subgraphs $Gmig_j$ and $Gmig_C$, through which this operator path π_{ij} traverses, and we select the upstream replicated operator of ec_j to be removed from the Fog node F_j. If this action improves the resulting end-to-end latency, we continue to remove the next upstream replicated operator, as long as the constraint $B \leq Bmax$ is satisfied. We stop applying $operatorMoveback()$ if the constraint $T \leq Tmax$ is satisfied. However, if the constraint is finally not satisfied or if removing a replicated operator does not improve the resulting end-to-end latency, we next apply the function $operatorMoveDown()$.

Operator move down $operatorMoveDown()$ processes similarly. Rather than removing the replicated operators from the Fog, it 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. The pseudo-code of the function

Algorithm 13: 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|z = 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
```

$operatorMoveDown()$ is presented in Algorithm 13.

6.4.1.2 Improve the overall resource usage cost

Aiming to improve the overall resource usage cost, we apply the function $edgeCutMove()$ (Algorithm 11, line 16) to identify all possible backward edge-cut moves [151]. We put the identified edge-cut moves and their ΔRU values in the set ΔRUset.

To apply the edge-cut moves (Algorithm 11, line 17-30), we sort the set ΔRUset in increasing order. Then, we pull from the top of the set the smallest ΔRU. If ΔRU is lower than 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 $\Delta\text{RU} \geq 0$. Like in SOO-H, TSOO-H updates at most once each data stream S_j with its best (lowest) ΔRU.

6.5 Experimental evaluation

We use iFogSim to simulate an Edge-Fog-Cloud architecture as well as several DSPA applications. Besides the TSOO-H and TSOO-SA algorithms, we also implement in our experimental

framework the following competitor methods:

Approach by Rizou et al. [74] This solution proposes an algorithm for distributing operators on a peer resource network. This algorithm considers a latency (i.e., propagation delay) space, where each host has a virtual position. It determines for each operator its optimal hosting node that minimises its network resource usage in the latency space, depending on the data stream rates between this operator and its neighbour operators. Starting from this solution, if the response time constraint is not satisfied, the algorithm tries moving each operator to the host that increases the network usage as little as possible, until satisfying the response time constraint. In particular, this approach can be mapped to TSOO-H, with the differences that it minimizes only the overall network resource usage cost (i.e. NRU). Then if the resulting solution does not satisfy the constraint $T \leq Tmax$ but it satisfies the constraint $B \leq Bmax$, in this case it attempts to satisfy the constraint $T \leq Tmax$ by applying $operatorMoveback()$ and $operatorMoveDown()$ like in TSOO algorithm, see Algorithm 11 (lines 9-13)).

TRCS (Time and Resource Constraint Satisfaction) We extend here the baseline approach RCS (cf. Algorithm 1). TRCS enhances pure IoT Cloud analytics by dynamically placing the operators between the Cloud and the Fog 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. Algorithm 14 presents the pseudo-code of TRCS.

More specifically, if the constraint $B \leq Bmax$ is satisfied while the constraint $T \leq Tmax$ is not satisfied, TRCS (similarly to TSOO-H) applies a search in the subgraphs $Gmig_j$ and $Gmig_C$ to select the operators to move from the Fog to the Cloud (or the inverse) in order to reduce T until the time constraint is satisfied, while keeping satisfied the constraint $B \leq Bmax$ and $B \geq Bmin$. This greedy search is performed with the functions $operatorMoveback()$ and $operatorMoveDown()$, which we described in Section 6.4.1.1. However in the case of TRCS, for each operator path, the function $operatorMoveback()$ will move an operator from the Fog to the Cloud only if the constraints $B \leq Bmax$ and $B \geq Bmin$ are satisfied and the resulting end-to-end latency is improved. In this respect, one should update Algorithm 12 (line 10) in order to include the constraint $B \geq Bmin$. Similarly, the function $operatorMoveDown()$ replicates an operator in the Fog only if the constraints $B \leq Bmax$ and $B \geq Bmin$ are satisfied and the resulting end-to-end latency is improved. We should also update Algorithm 13 (line 10) in order to add the constraint $B \geq Bmin$.

Algorithm 14: TRCS

Input: G, application graph
Input: S, set of S_j arriving to the Cloud
Input: B, Fog-to-Cloud network bandwidth usage
Input: T, DSPA application response time
Input: $Bmax$, Upper threshold for B
Input: $Bmin$, Lower threshold for B
Input: $Tmax$, Upper threshold for T
Input: Grep $\subseteq$ G, replicable subgraph in G
Input: Fog, set of Fog nodes F_j

```
if cmu_{F_j} > cm_{F_j} then
    AdjustEdgeCut()
if B > Bmax then
    ReplicateAndMigrateToFog()
else if B < Bmin then
    MigrateBackToCloud()
if T > Tmax & B ≤ Bmax then
    Set maxπ, sorted set of paths π_ij where Lπ_ij > Tmax
    operatorMoveBack(M, RM, G_dep, maxπ)
    if T > Tmax then
        operatorMoveDown(M, RM, G_dep, maxπ)
```

6.5.1 Experimental Setup

The experimental setup of data stream rates, DSPA application and Edge-Fog-Cloud architecture is the same as in the the previous Chapter (Chapter 5) with some differences that we point out in the following.

For the variability of the data stream rates arriving to the Fog nodes, we consider 10 values of M (number of IoT devices) selected randomly in the interval $[5000, 50000]$. In this respect, the total data rate reaching the Fog follows a sequence of 10 uniformly distributed values of M $\times 4KB/s$ repeated 15 times as depicted in Figure B.2. We feed this sequence to TSOO-H, TSOO-SA, TRCS and Rizou et al.

For the DSPA application, we use the same model of TLC application presented in Chapter 5 and depicted in Figure 5.9.

For the execution environment, we simulate the Edge-Fog-Cloud architecture as in Chapter 5, that includes 1 Cloud node, 10 Fog nodes and up to 50000 IoT devices at the Edge at the bottom. We use the tool Ether [165] to generate plausible (based on real data set) network configurations of the Edge-Fog-Cloud architecture. The distribution of the resulting network configurations follows the one used in [8]. For the computational resources, we simulate the Cloud node as an AWS VM instance of type m6g.xlarge [12]. At the Fog, we simulate ESXi virtual machines [166]. The MIPS evaluation of each resource node comes from [167]. Table 6.1 presents the configuration of the Edge-Fog-Cloud architecture.

Table 6.1: Network and computational resource parameters

Layer	Nodes	Prop. delay to up layer (ms)	Bandwidth to up layer (Mbps)	CPU (MIPS)	RAM (GB)
Cloud	1	-	-	35900	12
Fog	10	[100,300]	[100, 250]	[2400, 8150]	[1, 4]
Edge	[5000, 50000]	[10, 100]	[10, 50]	-	-

Finally for the threshold parameters, we set Bmax=125MB/s and Bmin=30MB/s. Given that the maximum propagation delay among all the network links is 300ms (Table 6.1), we set $Tmax$=500ms.

6.5.2 Evaluation results

We present in this section the evaluation results of TSOO-H against the competitor algorithms. We first discuss the results in terms of the resource usage cost achieved, then in terms of satisfaction of the resource usage constraint and response time constraint. Finally, we present the results in terms of scalability.

6.5.2.1 Resource usage cost

Figure 6.1 shows that TSOO-H finds the optimal RU, like TSOO-SA, whatever the data stream rates for up to M=30000 IoT devices and for $M \geq 45000$ IoT devices. For the other data stream rate values where M $\in$ [35000, 40000], TSOO-H approximates the optimal RU with an approximation error of only up to 5.14%, when comparing to TSOO-SA. Furthermore, TSOO-H outperforms TRCS, whatever the data stream rates, with a difference ratio respectively up to 29.23%. On the other hand, TSOO-H outperforms the approach by Rizou et al. for the data stream rate values where M $\in$ [35000, 40000] with a difference ratio up to 4.11%, while for the remaining data stream rate values Rizou et al. performs like TSOO-H. In particular, TSOO-H performs better than TRCS and Rizou et al. because TRCS does not aim to optimize the overall resource usage cost and Rizou et al. optimizes only the network resource usage cost.

6.5.2.2 Constraint satisfaction

We adopt the strategy of deploying a scheduling solution in all cases, the best one found or the least bad one. Hence, we apply constraint relaxation when an algorithm does not find a solution that satisfies all the constraints. In this section, we analyze how often the RU result achieved for each value of M satisfies or not the response time constraint ($T \leq Tmax$) and the Fog to Cloud bandwidth usage constraint ($B \leq Bmax$). To do so, we divide by 15 the number of times that the value of B (or T) does not satisfy the related constraint per each value of M (as we run

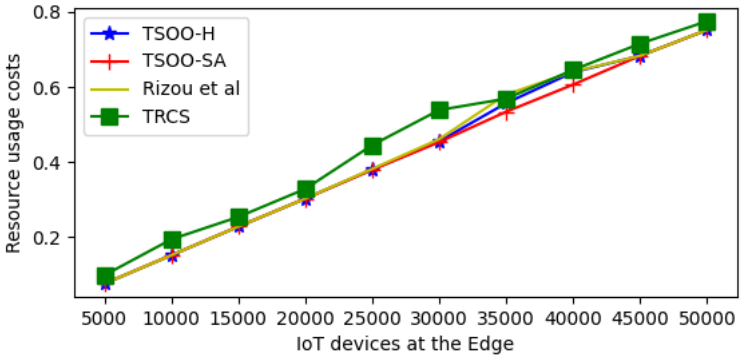


Figure 6.1: Overall resource usage cost

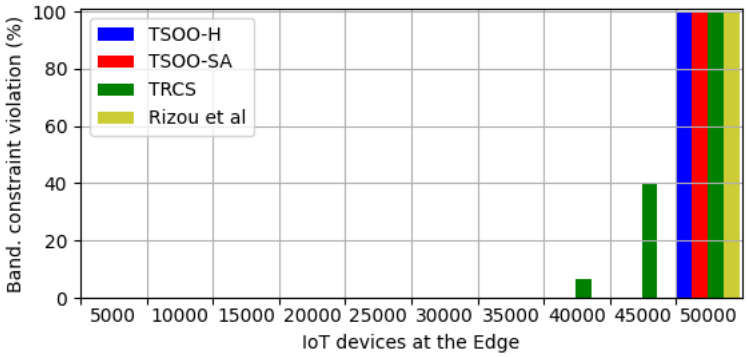


Figure 6.2: Violation rate of the Cloud bandwidth constraint

15 experiments per value of M). This gives the percentage of constraint violations per value of M.

Cloud bandwidth usage constraint. As depicted in Figure 6.2, TSOO-H satisfies this constraint whatever the data stream rates for up to M=45000 IoT devices. TSOO-H does not satisfy the constraint at the highest data stream rate value (M = 50000 IoT devices); it results in 100% of constraint violations. On the other hand, TSOO-SA and Rizou et al., same as TSOO-H, also fail to satisfy the cloud bandwidth constraint only at the highest data stream rate (M=50000) with 100% of constraint violations. At the same time, TRCS does not satisfy this constraint starting already at M = 40000 with about 7% of constraint violations, and reaching 100% of constraint violations at the highest data stream rate.

Response time constraint. As depicted in Figure 6.3, TSOO-H satisfies this constraint at lower and moderate data stream rates where $M \leq 35000$ IoT devices. For data stream rates where $M \geq 40000$, it may fail to fulfill this constraint with constraint violations that range between 73% and 100%. The approach by Rizou et al. performs similarly to TSOO-H in terms

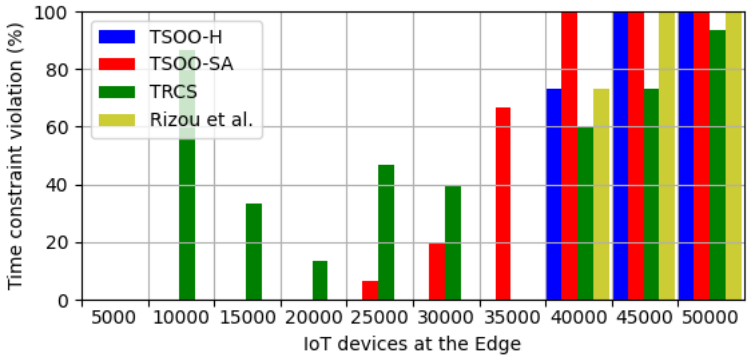


Figure 6.3: Violation rate of the response time constraint

of time constraint violations.

TSOO-SA satisfies the time constraint at lower data stream rates ($M \leq 20000$). Upon moderate or higher data stream rates, it may not satisfy this constraint with an increasing percentage of constraint violations (from 6.67% to 100%) as the data stream rate value increases. This means that in certain cases, relaxation of the time constraint was necessary in order to allow to TSOO-SA to find a solution, which was finally slightly better than the solution found by TSOO-H, as presented in Section 6.5.2.1. The results of TSOO-SA could possibly be improved with careful tuning of its parameters. However, this was not the focus of our work.

Finally, TRCS may fail to satisfy the time constraint even at low data stream rates (93.3% of constraint violations at $M = 10000$). This is due to the fact that TRCS prioritizes the satisfaction of the cloud bandwidth constraint. Nevertheless, TRCS performs better than all the other approaches at high data stream rates. It sometimes manages to find solutions that satisfy the time constraint where all the other approaches register 100% of constraint violations. However this result of TRCS does not lead to solutions without any constraint relaxation, as TRCS registers at these data stream rates more violations of the cloud bandwidth constraint than the other approaches (cf. Figure 6.2).

6.5.2.3 Scalability analysis

We assess the scalability of TSOO-H against TSOO-SA and Rizou et al. in terms of execution time, i.e., the time it takes for each algorithm to find the best operator placement. In this respect, we consider 6 different DSPA applications built based on the TLC application. Thus, App-1 has 5 operators (cf. Figure 5.9), App-2 has 6 operators, ..., and App-6 has 10 operators. We keep the same configuration of the Edge-Fog-Cloud architecture along with the parameter setting as presented in Section 6.5.1. We plot the results for low data stream rates where M=15000 and high data stream rates where M=40000.

At low data stream rates (cf. Figure 6.4 in log scale), the execution time increases with the

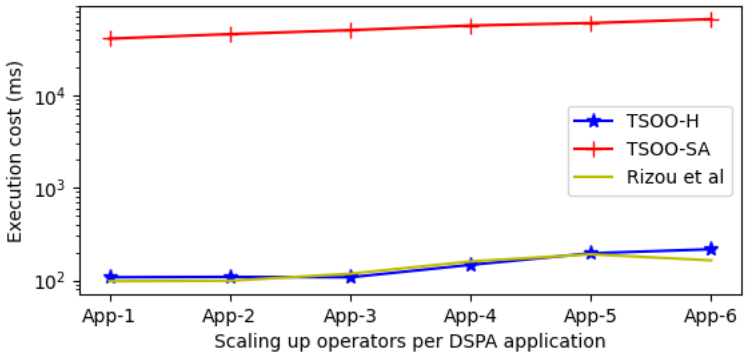


Figure 6.4: When M=15000 IoT devices

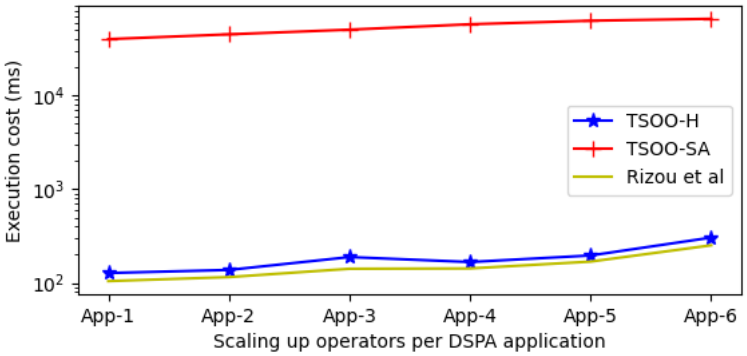


Figure 6.5: When M=40000 IoT devices

number of operators per DSPA application. TSOO-SA has much higher execution times than the other two algorithms. The execution cost of TSOO-H is on the same order of magnitude as the one of Rizou et al., with a difference ratio of up to 30% (TSOO-H being more costly).

At high data stream rates (cf. Figure 6.5 in log scale), the execution time similarly increases with the number of operators per DSPA application. TSOO-SA has still very high execution times; when comparing to TSOO-H, the latter reduces this execution time in the order of 99%. The execution cost of TSOO-H is higher than the one of Rizou et al. with a difference ratio of up to 51.37%.

6.6 Conclusion

In this chapter, we extended the problem of scheduling DSPA operators between the Cloud and the Fog in order to account for response time constraints of DSPA applications.

When evaluating the proposed algorithms, we can see that for lower data stream rates, TSOO-H is likely to find the optimal solution in terms of minimum overall resource usage cost and satisfaction of the problem constraints. However at higher data stream rates, TSOO-H may fail to find the optimal solution with an approximation error of up to 5.14%. Nevertheless, these solutions have lower rate of constraint violation of the problem constraint when comparing to TSOO-SA, TRCS and Rizou et al. Unless at highest data stream rates where all the proposed algorithms are likely to not satisfy the problem constraints.

Furthermore, we investigate also the scalability of the TSOO-H, we observed reduction in execution time of TSOO-H in the order of up to 99% when compared to TSOO-SA. Even though that Rizou et al. has lower execution time, TSOO-H provides the lower overall resource usage cost with lower lower constraint violation rates.

The current resource usage cost model fits for static deployment of DSPA application across the Edge-Fog-Cloud architecture. Static deployment of a DSPA application does not take into account the actual state of the Edge-Fog-Cloud resources. Whereas, it may require to dynamically re-schedule the current deployed DSPA application in reaction to the evolution of its workload since we assumed that DSPA application process data streams that come with different rates. Furthermore the Edge-Fog-Cloud resources can be used by other (DSPA) applications and hence, the available Edge-Fog-Cloud resource capacities can be dynamic.

Thus, in the next Chapter, we will improve the current resource usage cost model to fit for dynamic deployment of DSPA application. We will also extend the TSOO-H algorithm in order to adaptively re-schedule the current deplyment of DSPA application with respect to the dynamic environment (i.e. dynamic DSPA application workload and dynamic available Edge-Fog-Cloud resource capacities.

Chapter 7

Adaptive Scheduling of Continuous Operators

Contents

This chapter contains 25 pages.

7.1 Introduction

In the previous chapters, we studied how to obtain a scheduling of a DSPA application that takes into account the constraints in the computational resources of the Fog layer and the network resources between the the Fog and Cloud layers, as well as the DSPA application response time constraint.

Given that long running DSPA applications are characterized by a dynamic workload related to the varying IoT data stream rates [168], the algorithms that we proposed so far produce only

a static scheduling for each individual workload. This means that they do not take into account the previous deployment state of the DSPA application (except in the case of RCS/TRCS).

In this Chapter, we study how to adapt scheduling decisions to take into account the workload dynamics of the DSPA application, its deployment state, as well as, the dynamic available resource capacities of the Edge-Fog-Cloud architecture supporting the execution of several DSPA applications. Specifically, in this chapter we consider that the Edge-Fog-Cloud resources can be shared among several DSPA application.

The remainder of this Chapter is organized as follows. In Section 7.2, we extend the TSOO problem by introducing the dynamic operator placement problem. In Section 7.3, we propose the aTSOO-H algorithm for solving this problem. Finally, in Section 7.4 we thoroughly evaluate aTSOO-H.

7.2 Problem statement

For scheduling dynamically DSPA application across the Edge-Fog-Cloud architecture, we need to consider its current resource state in order to differentiate the usage of any individual resource across the Edge-Fog-Cloud architecture not only by its maximum resource capacity but also by its available resource capacity. In this respect, to assess the cost of using each computational or network resource, we use the dynamic weight version (see Formula (4.5)).

7.2.1 Computational and network resource usage costs

This chapter extends the TSOO problem in order to address the problem of adaptively scheduling the DSPA applications between the Fog and Cloud nodes. In this respect, for the computational resource usage cost, we use Formula (5.1) as in the TSOO problem we still focus on minimizing the usage of Fog computational resources. However rather than to statically calculate the weight, in this chapter we use the dynamic weight version to cope with the dynamic requirement of the environment. Thus, we use the dynamic weight version Formula (4.5) to calculate the weight factor of using a Fog node F_j at run-time:

$$W_{Fj} = 1 - \frac{cma_{Fj}}{cm_{Fj}} + \frac{cmu_{Fj}}{cm_{Fj}}, \tag{7.1}$$

Where, for a Fog node F_j, cm_{Fj} and cma_{Fj} are respectively the maximum and available computational resource capacities and cmu_{Fj} is the request of using the computational resources of a Fog node F_j.

Given the Formula (7.1) of the dynamic weight, given the overall Fog computational resource usage cost (cru) defined in Formula (5.1), the overall Fog computational resource usage cost becomes:

$$cru = \sum_{j=1}^{N}(cmu_{F_j} \cdot W_{Fj}) \equiv \sum_{j=1}^{N}(cmu_{F_j} \cdot (1 - \frac{cma_{Fj}}{cm_{Fj}} + \frac{cmu_{Fj}}{cm_{Fj}})) \tag{7.2}$$

Furthermore, for the network resource usage cost, we use Formula (5.2) as in the TSOO problem we still focus on minimzing the Fog to Cloud network resource usage. However we use the dynamic weight version to cope with the dynamic environment. In this case, the weight factor of using a network link from a Fog node F_j to a Cloud node C is as following:

$$W_{F_jC} = 1 - \frac{nba_{F_jC}}{nb_{F_jC}} + \frac{nbu_{F_jC}}{nb_{F_jC}} \tag{7.3}$$

where nb_{F_jC} denotes the maximum network bandwidth capacity, nba_{F_jC} the available network bandwidth capacity, and nbu_{F_jC} the request of using the available network bandwidth.

Then, by taking into account the weight factor defined in Formula (7.3), the overall Fog to Cloud network resource usage cost defined in Formula (5.2) becomes:

$$nru = c + \sum_{j=1}^{N}(nbu_{F_jC} \cdot W_{F_jC}) \cdot nd_{F_jC} \equiv c + \sum_{j=1}^{N}(nbu_{F_jC} \cdot (1 - \frac{nba_{F_jC}}{nb_{F_jC}} + \frac{nbu_{F_jC}}{nb_{F_jC}})) \cdot nd_{F_jC} \tag{7.4}$$

To normalize the computational resource usage cost, we devise *cur* by the sum of the maximum capacity of all the Fog nodes F_j (i.e. cm_{F_j}). In this respect the normalized form of *cru* defined in Formula (7.2) becomes:

$$CRU = \frac{cru}{cru_{max}} \text{ where } cru_{max} = \sum_{j=1}^{N} cm_{F_j} \tag{7.5}$$

Finally, 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 unit after the normalization. Then finally, 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 (7.4) becomes:

$$NRU = \frac{nru}{nru_{max}} \text{ where } nru_{max} = \sum_{j=1}^{N} nb_{F_jC} \tag{7.6}$$

7.2.2 Adaptive operator placement problem

Given the overall Fog computational resources usage cost, i.e., CRU and the overall Fog-to-Cloud network resource usage cost, i.e., NRU, the initial optimal placement $\mathcal{M}$ of an application graph *Gdep*, between the Fog nodes F_j and the Cloud node C should minimize both CRU and NRU

while satisfying the resource usage constraint and the response time constraint. As in Chapters 5 and 6, we consider the initial optimization of the TSOO problem as following:

$$\text{minimize} \quad RU = w_c \cdot CRU + w_n \cdot NRU \tag{7.7}$$

$$\text{Subject to:} \quad T \leq Tmax \tag{7.8}$$

$$B \leq Bmax \tag{7.9}$$

$$cmu_{Fj} \leq cma_{Fj} \ : \forall Fj, j = 1, \ldots, N, \tag{7.10}$$

Unlike in the previous Chapters where we constrained the usage of each Fog node F_j by its maximum capacities. In order to take into account the computational resource state of each Fog node F_j, in this Chapter we constrain the usage of each Fog node F_j by its available resource capacities (i.e. cma_{Fj}) (see Formula (7.10)).

By considering that the Edge-Fog-Cloud resources are shared among several DSPA application, the available computational and network resource capacities can be dynamic as long as the DSPA applications can be (re)deployed or removed on the fly. Moreover, the number and rate of individual data streams S_j produced from an IoT area A_j may vary according to the mobility patterns of IoT devices E_i [30]. Under these conditions, an optimal placement $\mathcal{M}$ statically defined may not anymore be a feasible solution to the TSOO problem. For this reason, we need to adapt the current operator placement $\mathcal{M}$ by rescheduling an already deployed application graph G_{dep} at run-time.

In essence we need to compute a new operator placement $\mathcal{M}$ that optimizes RU and satisfies the TSOO problem constraints by taking into account the current state of the Edge-Fog-Cloud resources.

Such adaptive scheduling of operator requires to take into account: (i) the cost of monitoring the current operator placement scheme, which may cause delay and network resource usage overhead; and (ii) the rescheduling cost that should take into account the number of replicated or removed operators. An efficient rescheduling strategy should have lower cost.

7.3 Proposed solution

To find an initial scheduling of DSPA application at deployment time we rely on the TSOO-H algorithm presented in Algorithm 11. However, we consider that the overall resource usage cost model takes into account both the available and maximum resource capacities. Moreover, the available resource capacities are used as upper bound of the resource usage constraints. Then, we propose the adaptive solution called aTSOO-H algorithm for evolving the DSPA application at run-time. Before detailing aTSOO-H, we first explain how we can monitor the execution of a deployed DSPA application between the Fog and Cloud nodes that processes dynamic data stream rate produced by IoT devices (Edge layer).

7.3.1 Monitoring the Distributed Execution of a DSPA application

DSPA engines such as Apache Flink [54] run several DSPA applications. A DSPA engine consists of Job Manager (JM) and multiple Task Managers (TMs) distributed across the nodes. By assuming that IoT data streams produced at the Edge are processed on Fog and Cloud nodes, the JM can be deployed on Cloud as it provides practically unlimited resources and a TM can be deployed on each Fog and Cloud nodes [8]. Then, the JM is responsible for planning the execution of operators by the mean of a scheduler (e.g. aTSOO-H) and to assign them to the TMs. Then, each TM executes the assigned operators.

We assume that the JM enabled with a global monitoring service while each TM with a local monitoring service. In this respect, at run time each TM continuously monitors metrics such as λ_x, μ_x of each operator O_x that constitute the sub-graph $Gmig_{n_j}$ deployed on each resource node n_j, $(n_j = F_j|C)$ along with the computational resource usage (i.e., cmu_{nj}) and the available computational resource capacity (i.e., cma_{nj}) on this node. Furthermore, it monitors for each individual Fog to Cloud network links, the values of the network bandwidth usage (i.e., nbu_{FjC}) and the current network delay (i.e., nd_{FjC}), see Formula (4.20) and Formula (4.21). Then, it sends at each fixed time interval t the average value of the monitored metrics to the JM instance in the Cloud.

Then, JM aggregates the reported values to calculate the current overall resources usage cost $RU(t_m)$ at each monitored time interval tm ($t \leq t_m$), the current overall Cloud bandwidth usage B, and the application response time T. It also calculates the current computational usage of the target DSPA application on each individual Fog node F_j (i.e., $cmu(t_m)$), as well as, the available computational resource (i.e., cma_{Fj}) of each individual DSPA application. Finally, it calculates the current network bandwidth usage of the target DSPA application on each individual Fog to Cloud network link (i.e., $nbu_{FjC}(t_m)$).

To decide whether $\mathcal{M}$ still satisfies our optimisation objective, we consider (i) RU_{max} an upper threshold of RU; and (ii) ϕ_k a term that penalizes the violation of any constraint k of the TSOO problem:

$$\phi_k = max(0, U_k - A_k) \tag{7.11}$$

where for each constraint k, U_k can be the value of $cmu_{Fj}(t_m)$, $nbu_{FjC}(t_m)$, B or T and A_k can be the value of cma_{F_j}, $Bmax$ or T_{max}. The current deployment $\mathcal{M}$ is acceptable under the following conditions:

$$RU(t_m) \leq RUmax, \tag{7.12}$$

$$\sum_{k=1}^{k_{max}} \phi_k = 0 \tag{7.13}$$

In this context, we need to search for a new operator placement $\mathcal{M}$ if at least one of the conditions (7.12) and (7.13) is not satisfied. However, defining statically the value of $RUmax$ does not guarantee to have RU bellow $RUmax$. In this respect $RUmax$ should be relative to RU. In this way rescheduling of the current $\mathcal{M}$ at time t_m is triggered when $RU(t_m)$ value exceeds or falls below $RUmax$; where $RUmax$ is equal to the RU value (optimal or best found) achieved with the last scheduling decision by more than a given relative threshold, (e.g., 10%). A small relative threshold will probably lead to more frequent rescheduling, depending also on how much and how fast the data stream rates evolve. This mechanism could also be complemented by setting a second, upper threshold for limiting the frequency of rescheduling execution.

7.3.2 aTSOO-H Algorithm

The scheduler takes the current operator placement $\mathcal{M}$ and the corresponding application graph G_{dep} as input, then it produces as output a new operator placement $\mathcal{M}$ and the resulting application graph G_{dep} to deploy so that the conditions defined in Formulas (7.12) and (7.13) are satisfied. The pseudo code of the algorithm aTSOO-H that we introduce to solve this problem is depicted in Algorithm 15.

aTSOO-H 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 ($cmu_Fj \leq cma_Fj$) (Algorithm 15, lines 2-7).

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 15, lines 9-10). Then, aTSOO-H builds the set $Sraw2$ that contains the data stream S_j that have been already migrated in the Fog and sorts them in decreasing order by their rates (Algorithm 15, line 11-12). 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 15, lines 14-18).

Then, if the constraint $T \leq Tmax$ is not satisfied, aTSOO-H searches in subgraphs $Gmig_j$ and $Gmig_C$ the operators to move from the Fog to the Cloud (or the inverse) in order to reduce T until the constraint $T \leq Tmax$ gets satisfied. This greedy search is performed with the functions $operatorMoveback()$ and $operatorMoveDown()$ (Algorithm 15, lines 18-22), which we describe

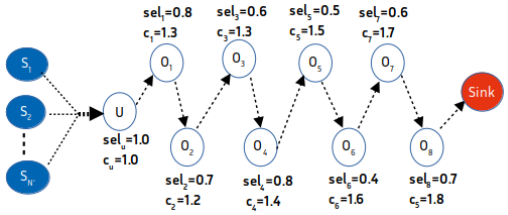
in Section 6.4.1.1. If both constraints $B \leq Bmax$ and $T \leq Tmax$ are finally satisfied, aTSOO-H algorithm applies a greedy search to further minimize RU while keeping satisfied the problem constraints (Algorithm 15, lines 23-24). This greedy search is described in Section 6.4.1.2.

It 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.

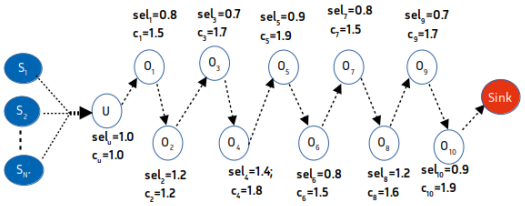
Algorithm 15: aTSOO-H

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

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



(a) DSPA application App-1 sharing the Edge-Fog-Cloud network resources



(b) DSPA application App-2 sharing the Edge-Fog-Cloud network resources

7.4 Experimental Evaluation

This section describes our experimental setup and the analysis of the obtained results. We use iFogSim to simulate two DSPA applications sharing the same Edge-Fog-Cloud architecture for processing the IoT data streams produced at the Edge.

We evaluate aTSOO-H against the algorithms TSOO-H and TRCS, described respectively in Section 6.4 and Section 6.5.1. For this comparison, we apply for all the algorithms our resource usage cost model version with dynamic weights, introduced in Section 4.3.1.2. As TSOO-H does not solve the TSOO problem dynamically, we run these algorithms from scratch each time a rescheduling is needed.

Furthermore, we compare aTSOO-H against the scheduling strategies proposed in [104], where the authors target a scheduling problem that is pretty similar to ours.

7.4.1 Experimental Setup

We present the two DSPA applications that we used in our experiments and the parameters that we applied to simulate the Edge-Fog-Cloud architecture. Then, we describe how we generate dynamic data streams produced by IoT devices to be processed by the two DSPA applications.

7.4.1.1 DSPA applications

As in Chapter 6, we consider the TLC application (New York City Taxi and Limousine Commission rides) [154]. From this application, we build two DSPA applications: App-1 and App-2, containing respectively 9 operators and 11 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 7.1a, 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 7.1b, we set the selectivity and cost of operators respectively from $[0.8, 1.2]$ and $[1.0, 1.9]$.

7.4.1.2 Simulated Edge-Fog-Cloud architecture

We again rely on iFogSim to simulate an Edge-Fog-Cloud architecture. In this architecture, we consider 1 Cloud node and 10 Fog nodes. On the other hand, 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.

For the computational and network resource parameters of our Edge-Fog-Cloud simulation, we consider that the Cloud node is an Amazon VM instance of type m6g.xlarge [12]. In the Fog, we simulate ESXi virtual machines [166]. The maximum reserved capacity of CPU and RAM of the Cloud node and the Fog nodes along with the parameters for the network topology in terms of propagation delay and network bandwidth capacities are the same as in Chapter 6. Table 7.1 presents the computational and network resource parameters used in these experiments.

Table 7.1: Network and computational resource parameters

Layer	**Nodes**	**Prop. delay to up layer (ms)**	**Bandwidth to up layer (Mbps)**	**CPU (MIPS)**	**RAM (GB)**
Cloud	1	-	-	35900	12
Fog	10	[100,300]	[100, 250]	[2400, 8150]	[1, 4]
Edge	[5000, 75000]	[10, 100]	[10, 50]	-	-

7.4.1.3 Dynamic IoT data stream rates

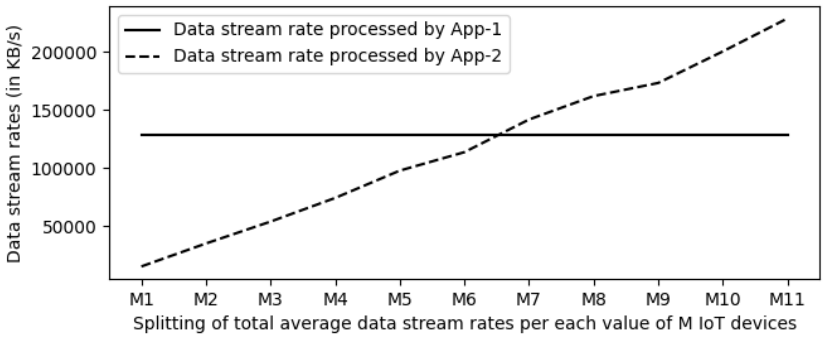
For the number of runs, like in Chapter 6, we statically simulate the variability of the data stream rates arriving to the Fog nodes by selecting randomly (uniform distribution) 11 values of M (number of IoT devices) in the interval $[5000, 75000]$, where each IoT device produces data at a rate of 6KB/s. Table 7.2 presents the resulting set of 11 values of M IoT devices. 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 Edge-Fog-Cloud 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 $m_{j1}(t)$ and $m_{j2}(t)$ so that each Fog node F_j receives data stream S_j splits between S_{j1} and S_{j2} with rate respectively $|S_{j1}| = m_{j1}(t) \times 6KB/s$ and $|S_{j2}| = m_{j2}(t) \times 6KB/s$ as the input of respectively App-1 and App-2.

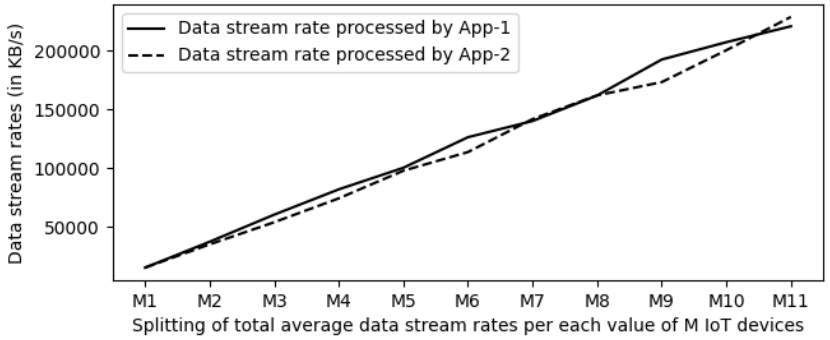
For the number of repetitions per run, we consider around 15 repetitions as in our previous set of experiments. In this respect, we repeat the splitting of each value of M per geographical area and per DSPA application 15 times. The total data rate reaching the Fog follows a sequence of 11 uniformly distributed values of M $\times 6KB/s$ repeated 15 times. We feed this sequence to aTSOO-H, TSOO-H, TRCS and to the state of the art solutions. We produce 15 results of T, B,

Table 7.2: Set of number of IoT devices (M) selected randomly in interval $[5000, 75000]$

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11
Value	5000	12000	19000	26000	33000	40000	47000	54000	61000	68000	75000



(a) Splitting of total average data stream rates in Scenario 1



(b) Splitting of total average data stream rates in Scenario 2

Figure 7.2: Distribution of data stream rate per applications and scenarios

RU, CRU, and NRU 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). This is achieved through two different scenarios that we explain in the following.

Scenario 1 In this scenario, the rates of data streams to be processed by App-1 are constant. Compared to the data stream rates of App-1, the rates of data streams to be processed by App-2 vary with values that are, for about half of them, lower than App-1, and for the other half, higher than App-1. In this respect, we fix the rate of input data streams of APP-1 approximately in the middle of possible values of M (i.e. M=40000 IoT devices in Table 7.2) and we leave the rest of the data streams for App-2. This is depicted in Figure 7.2a. The sequence of random data stream rates for both DSPA application are depicted in Figure B.3.

Scenario 2 In this scenario, we want to have close but not equal rates for the input data streams of App-1 and App-2. In this respect, we split the load of each M value between the two applications so that

App-1 processes the data streams produced by 55% of the M IoT devices, and App-2 processes the data streams produced by the other 45%. Thus, Figure 7.2b depicts the total input data stream rate per DSPA application for each value of M. The sequence in time of the data stream rates for either DSPA application is depicted in Figure B.4.

7.4.1.4 Setting threshold parameters

For each DSPA application, we set the threshold parameters Tmax, Bmax and Bmin as follows. $Bmax = 125MB/s$. Given that the maximum propagation delay among all the network links is 300ms (Table 7.1), we set $Tmax = 1000ms$, to allow some margin for operator latency and transmission delay. Finally, for TRCS we set $Bmin = 50MB/s$.

Given that we aim to evaluate the dynamic scheduling algorithm aTSOO-H and not the dynamic triggering mechanism described in Section 7.3.1. In this respect, we set the experiment to trigger aTSOO-H as well as TSOO-H at each change in data stream rates. In this way, we can evaluate all the scheduling algorithms on a common basis.

7.4.2 Evaluation results

In these evaluation results, we pay particular attention to the capability of the algorithms to solve the TSOO problem when scheduling two different DSPA applications one (App-1) after the other (App-2) across the Edge-Fog-Cloud resources, while taking into account the available resource capacities, thanks to the resource usage cost model version with dynamic weights. As evaluation metrics, we consider RU, T, B and the execution cost.

The execution cost encompasses the execution time and the 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 union operator) in the Fog or Cloud or (ii) deleted, i.e., they are 'migrated' back to the Cloud. Imporvement 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.

7.4.2.1 Analysis of resource usage costs and related constraint satisfaction rates

Scenario 1 For this scenario, the results achieved by each algorithm are presented in Table 7.3, Figure 7.3 and Figure 7.4.

Table 7.3: Results when scheduling App-1 in scenario 1 where data stream rate is static

	RU
aTSOO-H	0.285879722369758
TSOO-H	0.285879722369758
TRCS	0.301286945681742

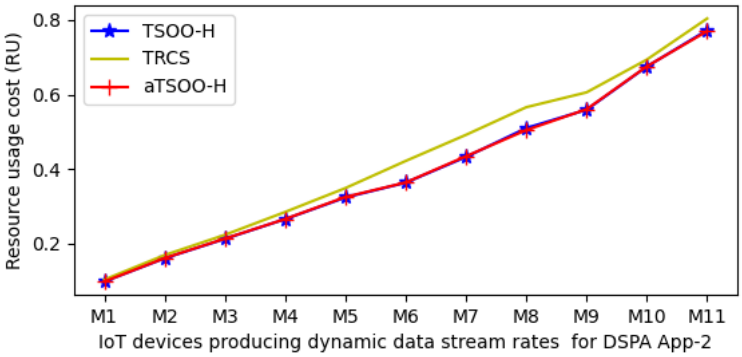
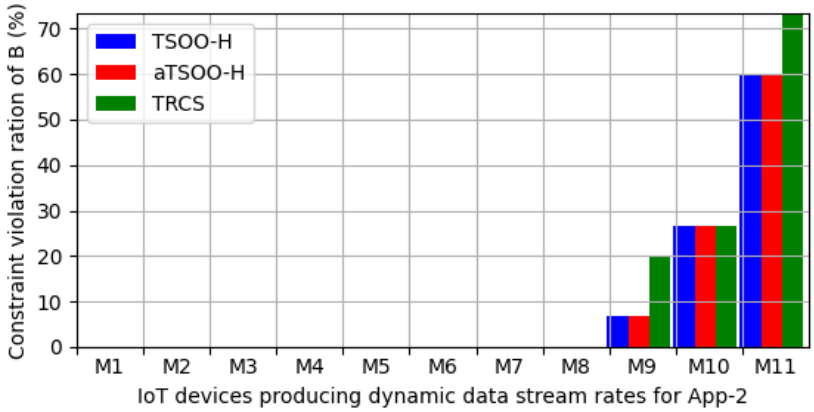


Figure 7.3: RU of App-2, when scheduling App-2 after App-1 in Scenario 1

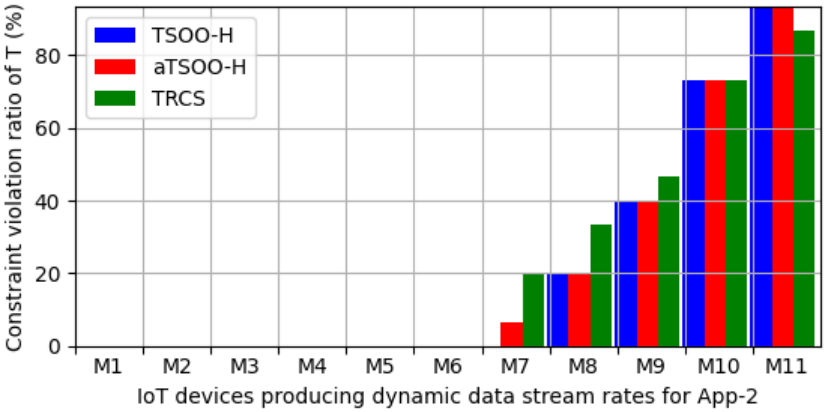
In this scenario the input data steam rate toward App-1 is static. In this respect, we present the overall resource usage cost of App-1 in Table 7.3. aTSOO-H like TSOO-H achieves optimal *RU* when comparing to TRCS, thanks to RUminCut algorithm. As introduced in Chapter 6, RUminCut attempts to select the edge-cut ec_j as the replication and migration point that minimizes directly the overall resource usage cost RU_j of each data stream S_j and Fog node F_j, then the resulting solution is optimal if all the TSOO problem constraints are satisfied. This is shown in Table 7.3, where aTSOO-H like TSOO-H has the lowest *RU* when comparing to TRCS.

In the second hand, when scheduling a DSPA application across the Edge-Fog-Cloud resources which are already used by other applications (e.g. App-1), the available resource capacities are reduced and hence the weight of using these resources are higher. As depicted in Figure 7.3, aTSOO-H provides lower *RU* when comparing to TRCS with a difference ratio of up to 15.88% from lower data stream rates until to the highest data stream rates (M1 to M11).

However, when comparing aTSOO-H against TSOO-H, we notice that TSOO-H provides optimal operator placement at lower data stream rates (i.e. M1 to M3). For the other data stream rates, TSOO-H approximates the optimal operator placement. Whence, TSOO-H outperforms aTSOO-H in terms of *RU* with a small difference of up to 0.0045 for all the data stream rates, except for the moderate data stream rate (i.e. M8) where aTSOO-H approximate the optimal operator placement than TSOO-H with a small difference of 0.01. At each time TSOO-H is triggered, the algorithm is executed from scratch without taking into account the



(a) Cloud bandwidth usage

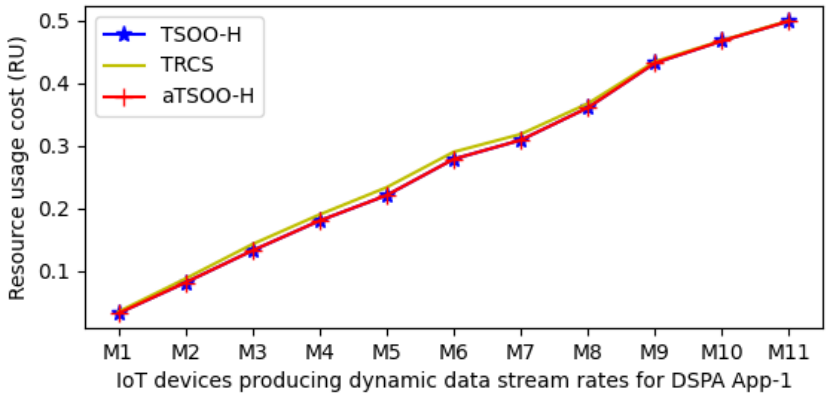


(b) Response time

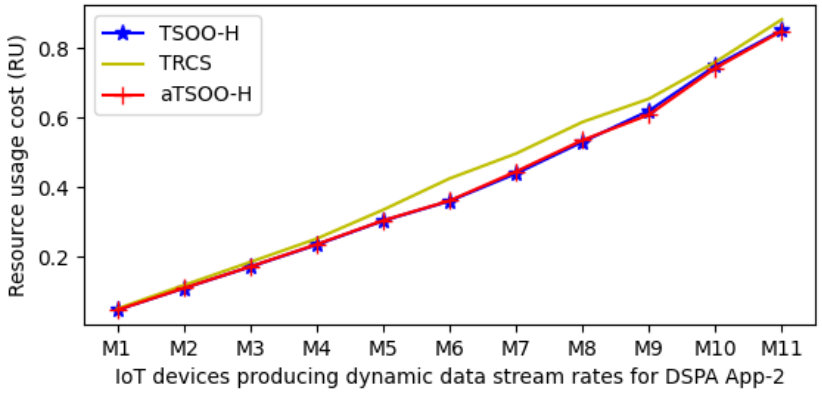
Figure 7.4: Constraint violation rates of App-2 in Scenario 1

current operator placement, and hence TSOO-H has higher probability to identify the operator placement which approximates the most the optimal *RU* (see Chapter 6). On the other hand, when aTSOO-H is triggered, it considers the current operator placement from which it produces the new operator placement that minimizes *RU* and satisfies the TSOO problem constraint. In this respect aTSOO-H is more likely to produce *RU* which is a bit higher than the one of TSOO-H (except for M8).

When analyzing the constraint violation rate of each algorithm, we observe that all the algorithms satisfy the TSOO problem constraint when deploying the App-1, that why we omit to present the related plots. However, when the available capacities of Edge-Fog-Cloud resources are reduced due to the workload of App-1 initially deployed, when scheduling App-2 the constraint violation rate is increasing with the data stream rates. For instance, Figure 7.4a, shows that for TRCS the violation rate of the constraint $B \leq Bmax$ is already 20% at the data stream rates produced by IoT devices of M9 value and it keeps increasing until to reach 100% at the highest



(a) RU of APP-1



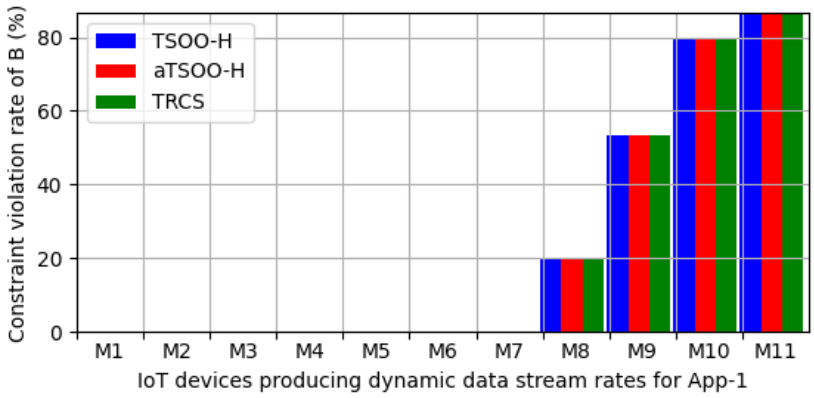
(b) RU of APP-2 w.r.t the load of App-1

Figure 7.5: Overall resource usage costs when scheduling App-1 and App-2 in Scenario 2

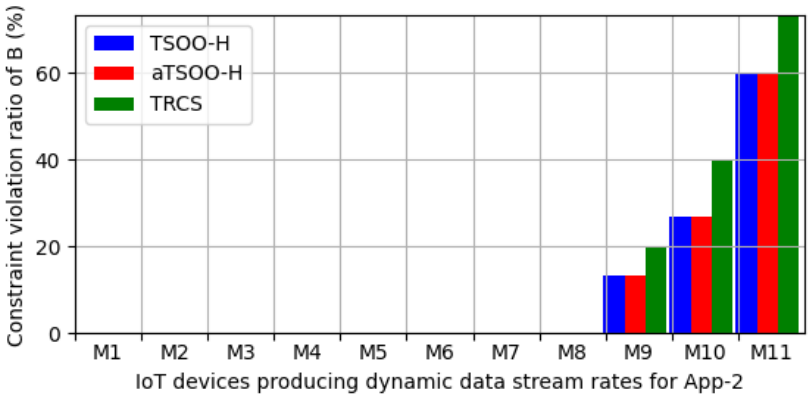
data stream rate produced by IoT devices of M11 value. While for aTSOO-H and TSOO the violation rate of this constraint is lower, specifically 6.67% at M8 and 6.67% at M9 and it keeps increasing however without reaching 100%.

Last but not least, Figure 7.4b shows that TRCS has the highest violation rate of the constraint $T \leq Tmax$ from M7 to M9 with respectively 20% to 46%. However it has the same violation rate as TSOO-H and aTSOO-H at M10 and it gets even lower than the latter at the highest data stream rate (i.e. $M11$) with 73.3% while they have the same highest violation rate of 93.3%.

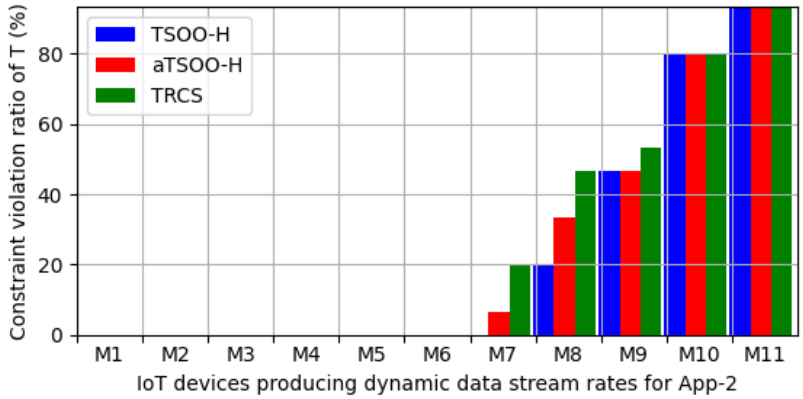
Scenario 2 We consider dynamic data loads for both App-1 and App-2 where App-1 is initially deployed across the Edge-Fog-Cloud architecture, then the App-2 is deployed when App-1 is processing data stream. Thus, both App-1 and App-2 have to share the Edge-Fog-Cloud resources to process the dynamic data stream rate produced at the Edge. The results achieved in this scenario are presented in Figure 7.5 and 7.6.



(a) Cloud bandwidth constraint for APP-1



(b) Cloud bandwidth constraint for APP-2



(c) Response time constraint for App-2

Figure 7.6: Constraint violation rates when scheduling App-1 and App-2 in Scenario 2

The plots of the overall resource usage cost (RU) are steeper as depicted in Figure 7.5a. In this respect, 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

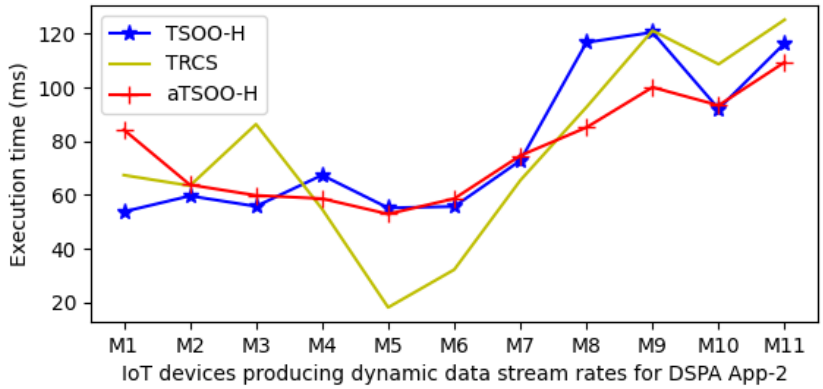
algorithm, and it approximates the optimal solution at the highest data stream rates (i.e. $M11$). Since we run TSOO-H from scratch at each time the rescheduling is triggered, it is more obvious to observe such performance of TSOO-H as long as we have shown in Chapter 6 that TSOO-H is more likely to identify the optimal (or near optimal) solution. Unlike TSOO-H, aTSOO-H is executed from scratch only for the first data stream rates in the random sequence of data stream rates where it produces also the optimal solution. For the following data stream rates, aTSOO-H adapts the current operator placement solution in order to provide an (near) optimal operator placement solution with respect to the actual workload. In this respect, aTSOO-H takes the advantage of the abundant available resources capacities across the Edge-Fog-Cloud 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 7.5b shows that TSOO-H has the lowest RU among all algorithms for the data stream rates produced from $M1 to M8$ IoT devices. However the difference ratio is very small this is due to fact that the input data stream rates are closer for the two DSPA applications. 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 algorithm 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%.

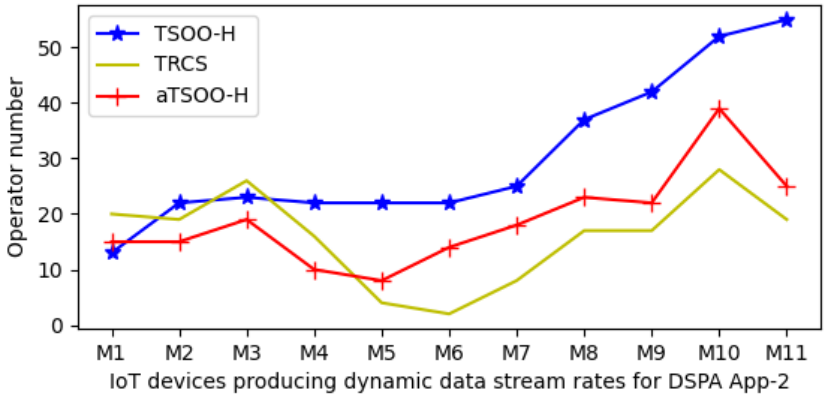
Regarding the TSOO problem constraints, all the algorithms satisfy the constraint $T \leq Tmax$ when scheduling App-1 at any data stream rates that why we omit to put the related plots. Even though that TSOO-H and aTSOO-H provide the lowest RU, like TRCS they are more likely to fail to satisfy the constraint $B \leq Bmax$ with an increasing probability as the data stream rates are increasing from M8 to M11 spanning from 20% to 86.67% of constraint violation rate, see Figure 7.6a.

For App-2, we observe that TRCS is likely to not satisfy the constraint $B \leq Bmax$. As depicted in Figure 7.6b, both aTSOO-H, TSOO-H and TRCS start failing to satisfy this constraint at higher data stream rates (i.e. $M9$ to $M11$) with hover 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%.

For the real time response constraint, Figure 7.6c shows that all the algorithms start failing



(a) Execution time



(b) rescheduling cost

Figure 7.7: Execution cost of App-2 in scenario 1

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.

7.4.2.2 Analysis of execution cost

As introduced above, we consider the execution cost of the algorithms in terms of the execution time and the rescheduling cost. In this respect, we purposely not put the plot of execution cost of App-1, as we consider constant data stream rate and consequently the initial scheduling solution identified by each of the algorithms is sufficient.

For App-2, the achieved results are depicted in Figure 7.7, Figure 7.8 and Figure 7.9. In these figures, we plot the average execution times and rescheduling costs of the 15 random executions of each value of M IoT devices.

Scenario 1 Figure 7.7a shows the execution times of the algorithms in scenario 1 when scheduling App-2. At lower data stream rates (i.e. M1 to M3), TSOO-H has lower execution time when

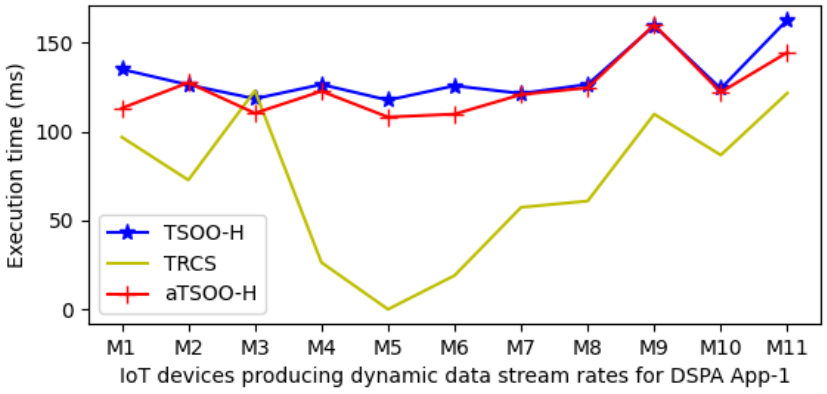
comparing to both aTSOO-H and TRCS with a difference ratio of respectively up to 56.63% and 54.71%. In this case, TSOO-H is executed in the favorite case of applying only RUminCut to find directly the optimal solution. From the next lower to the highest data data stream rate (i.e. M4 to M11), we observe that aTSOO-H has lower execution time than TSOO-H for 6 data points and the rest of 3 data points, the execution times are closer. In this case, TSOO-H relies on the costly greedy search part of the algorithm. Thanks to its adaptive approach, aTSOO-H achieves to identify the scheduling solution faster than TSOO-H. On the other hand, TRCS has the lowest execution time for the data stream rates produced from M4 to M7, in this setting the search for identifying the operators to replicate on Fog or to move back in Cloud in order to meet the TSOO problem constraint is reduced that may involve fewer Fog nodes. However at high data stream rates (i.e. M8 to M11), TRCS has highest execution time except for M10 where it performs costly like TSOO-H. In this case TRCS expands the search space that may involve all the Fog nodes and hence high execution time.

When analyzing the rescheduling cost, we can see in Figure 7.7b that at moderate and higher data stream rates (i.e., M5 to M11) TSOO-H replicates more operator in order to address the change in the DSPA application workload thus imposing high reconfiguration cost of the DSPA application The rescheduling cost of TRCS is lower however the resulting RU is higher than the one of aTSOO-H. In contrast, aTSOO-H reconfigures the current operator placement to a new one by replicating or/and removing only a minimal number of operator. Thus, aTSOO-H has lowest rescheduling cost than TSOO-H.

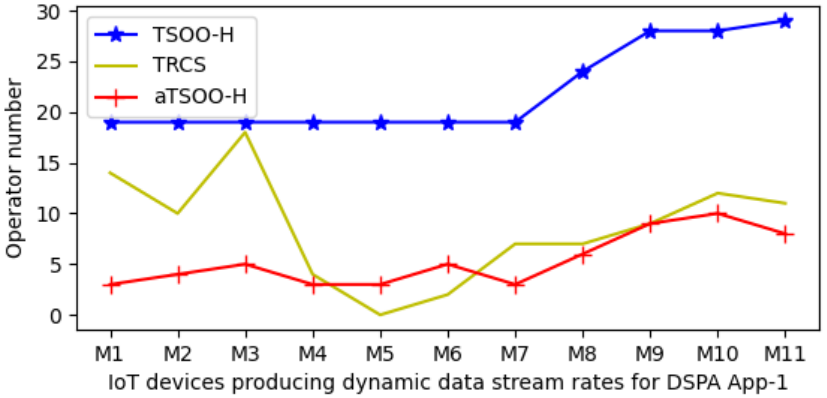
Scenario 2 In scenario 2 when scheduling App-1, Figure 7.8a shows that TSOO-H has the highest execution times from the lowest to the highest data stream rates (i.e. M1 to M11) except at M3 and M9 where aTSOO-H equals the high execution cost of TSOO-H. Even thought that TSOO-H is executing in the best 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.

In terms of the rescheduling cost, Figure 7.8b shows that the rescheduling cost of aTSOO-H is the lowest among all the algorithms except for $M5$ and $M6$ where TRCS provides the lowest rescheduling cost. Thanks to the adaptive approach of aTSOO-H that replicates or/and removes only the operator that are likely to solve the TSOO problem. 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 replicates 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.

The execution cost when scheduling App-2 after App-1 follows the same pattern as the exe-



(a) Execution time



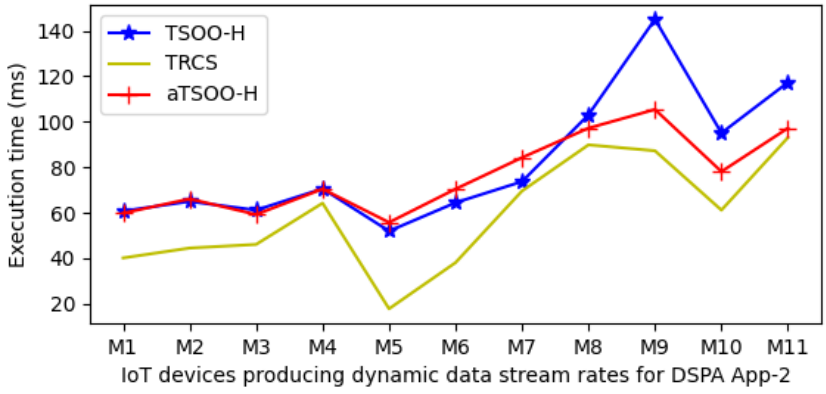
(b) Rescheduling cost

Figure 7.8: Execution cost of App-1 in scenario 2

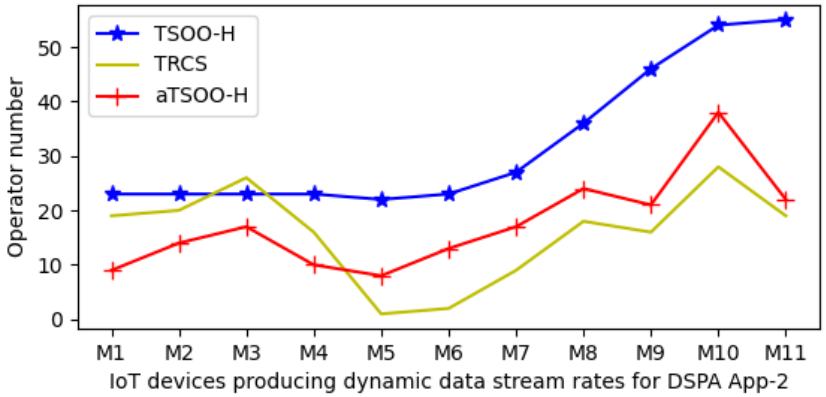
cution cost when scheduling App-1. However, in this case available resource capacities between the Fog and Cloud are not anymore abundant. Hence, by taking into account the actual available resource capacities and the application workload, the execution times of TSOO-H and aTSOO-H are still high than those TRCS (see Figure 7.9a). In terms of rescheduling cost 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 TSOO-H (see Figure 7.9b).

7.4.3 Comparison with state of the art solutions

As introduced in Chapter 3, Rzepka et al. [104] address the problem of scheduling on the fly several DSPA applications sharing the Edge-Fog-Cloud resources. The objective is to maximize the number of successfully deployed DSPA applications while limiting the WAN resources usage for transmitting data streams on the Fog to Cloud WAN links and using efficiently the Fog computational resources. However they distinguish between DSPA applications that have sink



(a) Execution time



(b) Rescheduling cost

Figure 7.9: Execution cost of App-2 in scenario 2

in the Cloud and those have sink in the Fog which are categorized respectively between non response time-critical and strict response time requirements applications. To solve the problem, they propose scheduling strategies (not optimization algorithms) designed based on the following insights:

- limite the WAN resources usage for transmitting data stream on the Fog to Cloud WAN links;
- efficiently use the Fog computational resources so that it can (also) be used by several (DSPA) applications;
- save the Fog computational resources for DSPA applications with strict latency requirements and use the Cloud for non time-critical DSPA applications.

Among all the scheduling strategies of [104], we observe that only FogOnly strategy can be applied directly in order to solve the TSOO problem. FogOnly deploys in the Fog the overall

DSPA application even if it has the sink in the Cloud in order to maximize the Fog resource usage. When applying the rest of the strategies to the TSOO problem, it falls that the overall DSPA application is deployed in the Cloud as in this thesis we assumed that each DSPA application has the sink in the Cloud. Deploying the overall DSPA application in the Cloud is the solution we want to avoid as it may bring network congestion and high network delays.

In this respect, we inspire from their insights to build another scheduling strategy called Fog Cloud Interplay (FCInterplay) to solve the TSOO problem. In this respect, given the data streams S_j, given the application graph G, FCInterplay attempts to solve the TSOO problem as follows:

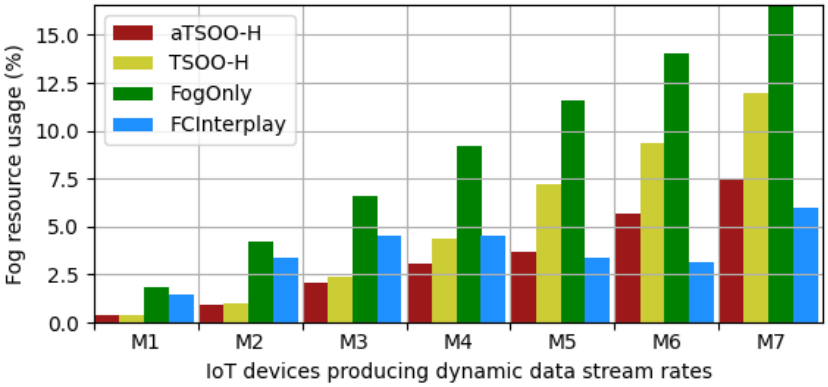
- Send directly a data stream S_j to be processed in the Cloud if the resulting end-to-end operator path latency can not satisfy in any way the response time constraint. This is to avoid wasting the Fog computational resources. However if only the Cloud bandwidth usage constraint is satisfied.
- A data 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.
- 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.

7.4.3.1 Evaluation results

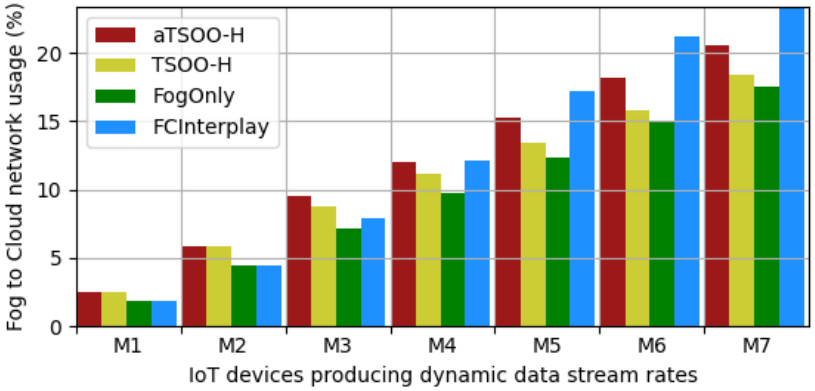
We compare aTSOO-H and TSOO-H with the FogOnly algorithm proposed in [104] and the FCInterplay algorithm inspired from [104]. As a reminder, FogOnly deploys the whole DSPA application in the Fog in order to maximize the Fog computational resource usage under the constraint of available resource capacity. However if the computational resources at the Fog are not sufficient, the request of deploying the DSPA application is rejected [104].

We use the same experimental setting parameters introduced earlier in this chapter. 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.

Then, to compare the identified algorithms we consider the following metric used in [104]: (i) the overall Fog to Cloud bandwidth usage ratio; (ii) the overall Fog computational resource usage ratio; and (iii) the DSPA application deployment success rate. The latter is calculated



(a) Fog computational resource usage ratio



(b) Fog to Cloud bandwidth usage ratio

Figure 7.10: Computational and network utilization ratio

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 [104]. Given the simulation of the dynamic data stream rates introduced in Section 7.4.1, we consider a 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 per each of the 11 (M) values of IoT devices. Hence, the deployment success rate is calculated on the basis of 30 requests per each M value of IoT devices. In the following we present the evaluation results of only scenario 2. These results are depicted in Figure 7.10 and Figure 7.11

Fog computational resource usage ratio Figure 7.10a 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. $L\pi_{ij} \leq Tmax$). 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 TSOO-H (and aTSOO-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. $L\pi_{ij} \leq Tmax$). 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.

While aTSOO-H and TSOO-H try to jointly optimize the usage of the Fog resources and Fog to Cloud network resources.

Fog to Cloud wide area bandwidth usage ratio At lowest data stream rates (i.e., M1 and M2), Figure 7.10a 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 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 as 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 fact that aTSOO-H and TSOO-H aim to jointly minimize the Fog resources and Fog to Cloud network resources.

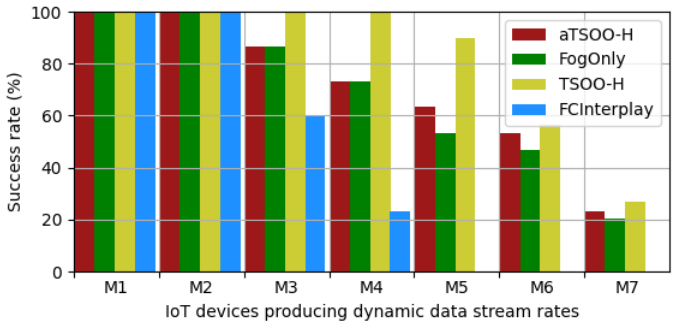


Figure 7.11: Deployment success rate in scenario 2

DSPA Application deployment success rate 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 FCInetraplay 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. However the operator processing times are higher on the Fog which impacts on the response time constraint. 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. We believe that the dynamic optimization approach of aTSOO-H that takes into account the current operator placement in order to produce a new operator placement does not enable aTSOO-H to converge to the best possible scheduling solution even if it provides lower execution cost.

7.5 Conclusion

In this chapter, we introduced the dynamic version of the TSOO problem, the problem of continuously scheduling DSPA application between the Fog and Cloud nodes in synergy with the change in the data stream rates and the available Edge-Fog-Cloud resource capacities. The objective was to jointly optimize the Fog computational resource usage and Fog to Cloud network resource usage while satisfying the real time response constraint of DSPA application.

In this respect, we use the version of the resource usage cost model with dynamic weights in order to take into account both maximum and available resource capacities when deploying

DSPA application across the Edge-Fog-Cloud resources. Then we propose aTSOO-H algorithm that adaptively schedules DSPA application by taking into account its current deployment state in order to solve the dynamic version of the TSOO problem.

aTSOO-H exhibits significant advantages when compared to TSOO-H and TRCS. In particular, aTSOO-H approximates better the optimal solution of the TSOO problem when comparing to TRCS. It worth noting that TSOO-H approximates better than aTSOO-H the optimal solution, however aTSOO-H has lower execution cost in terms of algorithm execution time and the number of operators replicated or removed at each rescheduling of the current deployment of DSPA application.

Lastly, we evaluate aTSOO-H against FogOnly from the recent related work and FCInterplay inspired from the related work. It shows that aTSOO-H has better performance than FogOnly and FCInterplay in terms of optimizing resource usage and successfully deploying DSPA application. In particular aTSOO-H has 100% of success rate when deploying DSPA application at lower data stream rates and the success rates is decreasing with the increasing of the data stream rates while never reach 0%. However this success rate remains higher than those of FogOnly and FCInterplay. It worth noting that TSOO-H outperforms also aTSOO-H in terms of DSPA application deployment success rate.

Chapter 8

Conclusions

Contents

8.1 Summary of Contributions

Data stream processing and analytics (DSPA) engines have been originally designed to run on centralized environments (e.g., Cloud) featuring high computational resource capacities. With the exponential growth of the Internet of Things (IoT), the quality of service (QoS) of DSPA applications is challenged by the need to systematically transmit data streams from IoT devices to the Cloud.

In this respect, processing data streams at the IoT network edge emerges as a promising solution for addressing congestion issues on long-distance communication network links. However, to this end we need to deploy DSPA applications on Edge/Fog nodes that come with non-trivial constraints in terms of heterogeneous, shareable and limited computational resources. In respect, a thorough allocation of Edge/Fog resources becomes crucial in order to meet QoS requirements of DSPA applications over dynamic IoT data streams. In this thesis, we leverage the computational resources available in the Edge/Fog layers to distribute DSPA computations performed in the Cloud. Hence, only partially processed IoT data streams at the Edge/Fog need to be finally send to the Cloud for further analysis.

In a nutshell, we address in this thesis the main challenges of distributing DSPA applications between the Fog and Cloud resources. As Fog computational resources and wide area network resources to reach the Cloud may exhibit a workload fluctuation that could impair on DSPA application response time constraints, we consider both static and dynamic scheduling approaches. These fluctuations are due to the fact that computational and network resources can be shared among several applications but also to the dynamics of the IoT data stream rates according to spatio-temporal patterns. More precisely, we are answering three particular questions: (i) How can we model the computational and network resources of Edge-Fog-Cloud nodes and the continuous operations of DSPA applications? (ii) How can we estimate the usage of resources allocated to the execution of a DSPA application given that Edge-Fog-Cloud nodes are highly heterogeneous and could be shared by different applications? and (iii) How can we afford an adaptive scheduling of DSPA applications in the Edge-Fog-Cloud continuum given the resource fluctuation at the Fog and Cloud layers as well the fluctuation of data stream rates from the Edge layer to the Fog layer ?

In the first contribution, we proposed abstraction models of the DSPA application and Edge-Fog-Cloud architecture. The DSPA application is abstracted as a directed acyclic graph of operators based on which we propose a model for estimating the resource required by each operator in terms of CPU/RAM to process its input data stream and network bandwidth for sending the processed data streams on the wire. In this respect, we consider the operator selectivity, the operator window type and the operator cost. The Edge-Fog-Cloud architecture is abstracted as a wide area resource network by specifying for each resource its available and maximum capacities. Based on the two abstractions, we build a holistic resource usage cost model for replicating and placing operators of DSPA applications from the Cloud to Fog resources, by distinguishing static and dynamic weighting the required computational and network resource usage of each operator respectively in the case of static and dynamic deployment of DSPA application. To cope with the response time constraint, we further propose a response time model that takes into account the network delay of sending a data stream on each individual wide area network link and the time required per continuous operator to process its input data streams. For the latter case, we model each operator as a queuing system.

In the second contribution, we exploited the above models to propose resource aware and time aware scheduling strategies for statically distributing a DSPA application between the Fog and the Cloud nodes by assuming that the target Edge-Fog-Cloud resources are dedicated to a single application. In this respect, we consider the version of the resource usage cost model with static weights. The static weights distinguish the usage a each resource by the inverse of its maximum capacity. Based on this resource usage cost model version, we first proposed resource aware scheduling algorithms, namely RCS, SOO-CPLEX and SOO-H. RCS is a resource

constraint satisfaction approach that dynamically evolves the DSPA application between the Fog and the Cloud, aiming to satisfy the computational and network resource requirement by using as less as possible the Fog computational resources. SOO-CPLEX is a mathematical optimization based approach that aimed at an optimal overall resource usage cost due to an optimal trade-off between the Fog computational resource usage cost and the Fog-to-Cloud network resource usage cost. Finally, SOO-H is a heuristic based approach that efficiently achieved the optimal overall resource usage cost in the best case like SOO-CPLEX, while it approximated in the worst case in a time efficient way the overall resource usage cost, with a small approximation error when compared to SOO-CPLEX.

In the third contribution, we proposed a time and resource aware scheduling strategy. In this respect, we extended SOO-H to TSOO-H to account for response time constraints. Experimental results showed that TSOO-H is scalable and time efficient when comparing to state of the art solutions. It approximates the optimal solution, while managing the trade-off between the usage costs of Fog computational resources and Fog-to-Cloud network resources and satisfying the response time constraint.

In the fourth contribution, we used the version of the resource usage cost model with dynamic weights to account for dynamic deployment of DSPA application on the Edge-Fog-Cloud resources that are shared among several applications. In this way the dynamic weights distinguish the usage of each Edge-Fog-Cloud resource by take into account not only its maximum resource capacity but also its available resource capacity. We account also for the fact that the dynamic data stream rates produced by IoT devices require dynamic and lightweight scheduling solutions. Hence, we proposed a monitoring framework for analyzing the workload of a DSPA application deployed between the Fog and Cloud resources and triggering an adaptive scheduling of the current operator placement whenever it is necessary. To this end, we proposed aTSOO-H, which, adaptively and with lower execution cost, reschedules the current operator placement of a DSPA application in order to minimize the overall resource usage and satisfy the response time constraint.

8.2 Future Research Directions

This work opens new challenges for future research that we briefly detail in the sequel.

8.2.1 Integrating with existing DSPA engines

To integrate our proposed scheduling solutions with widely-used DSPA engines (e.g., Apache Storm [53], Apache Kafka [55], Apache Flink [54]), we need to understand the architectural characteristics of the DSPA engine to be used.

For instance, when a DSPA application is submitted to the JobManager of Apache flink, the latter converts this application as JobGraphs. Then, the JobManager will call a schedule method that invokes an ExecutionGraph to subsequently call scheduleForExecution. This triggers a task allocation method for each of the vertices in the ExecutionJobVertex (a collection of vertices in JobGraphs). So, basically the idea is to override the task allocation algorithm with the scheduling strategies that we propose in this thesis. To re-schedule a running application, the idea could be to generate and maintain multiple plans of JobGraphs for each submitted DSPA application. These plans will be considered when evaluating a running DSPA application.

However, it is worth noting that the popular DSPA engines enumerated above are designed to run on resource-rich environments, while the Edge/Fog computing environment may come with constrained computational resources. Thus, effective deployment of DSPA engines in the Edge/Fog environment requires lightweight DSPA engines such as NebulaStream [65]. We expect this direction to be further explored as stable Edge/Fog oriented DSPA engines are still not yet established [20].

8.2.2 Generalization of the DSPA application type

In this thesis, we focused on the problem of scheduling a DSPA application in the Edge-Fog-Cloud continuum where the DSPA application is initially deployed in the Cloud and it has its sink there. One of the recent works that we surveyed proposed interplay strategies between the Fog and the Cloud for scheduling DSPA applications that have their sink either in the Cloud or in the Fog [104]. One of the insights drawn in [104] is that the best scheduling strategy is to favor deploying a DSPA application entirely in the Cloud or entirely in the Fog if it has respectively its sink in the Cloud or in the Fog. Actually, we compared our solutions with a strategy that we implemented inspired from [104] in Section 7.4.3. A future extension of our work is to consider not only DSPA applications that have their sinks in the Cloud, but also DSPA applications that have their sink in the Fog or in both the Fog and the Cloud. In the latter cases, a scheduling algorithm should possibly also take into account data streams in the direction Cloud to Fog.

8.2.3 Explore mobile Edge resources

In this thesis, we consider only Fog and Cloud computational resources, while IoT devices at the Edge come with non negligible processing capacities. However, extending the processing of data streams at the Edge layer possibly means dealing with mobile nodes, which can have a disruptive impact on the whole DSPA application. In this context, some efforts have been made to tackle the challenges associated with deploying DSPA applications on mobile IoT devices at the Edge. To cite some of them, O'Keeffe et al. [169] design an edge-based IoT data processing approach that leverages the computing abilities of multiple IoT devices in order to process data streams in parallel and to account for the unreliable network conditions in wireless networks. Chao et

al. [170] propose a solution to improve the resilience of DSPA applications deployed on mobile Edge resources interconnected via a wireless network in which the communication quality is severely affected by the environment condition, signal attenuation, channel contention. On the other hand, the upcoming of 5G network technologies [10, 11] enables increased network bandwidth capacity, high connectivity density of IoT devices, and, more importantly, highly reliable communication network links. In spite of these efforts, there are still challenges to address. We believe that a future exploration should pay particular attention to resource churning caused by the mobility of IoT devices and to energy consumption of mobile devices, in order to enhance the resilience of DSPA applications deployed on Edge resources.

8.2.4 Leverage Machine Learning Techniques

Applying machine learning (ML) techniques to the problem of scheduling DSPA applications in the Edge-Fog-Cloud continuum can be useful in various ways. In this respect, our threshold based monitoring approach can be enhanced by using ML techniques such as linear regression in order to predict in the near future the status of the resource usage and performance of a DSPA application, based on which the rescheduling of the current operator placement can be proactively triggered if necessary by taking into account the DSPA application characteristics, how the data stream rate evolve and also how much time it takes to deploy a new scheduling [8]

Furthermore, ML techniques such as reinforcement learning (RL) can be used in order to evolve the scheduling of a DSPA application in synergy with the evolution of available resources and data stream rates [115].

On the other hand, training ML models can be resource greedy, while the Edge/Fog layers may come with limited computational resources. Thus, it is necessary to investigate the use of Edge/Fog resource aware machine learning models, like in Federated Learning.

8.2.5 Privacy and security

Today IoT devices collect and produce data that are useful to consumers, businesses and public sector policy-making. In this context, privacy issues naturally arise, as these devices can collect and transmit personal data, from which insights about an individual's behavior, health or relationships can be inferred. On the other hand, existing security techniques, tools and products may not be easily deployable on IoT devices because of the variety of hardware platforms and limited computational resource capacity. Although processing data at the network edge through Edge/Fog computing can enforce data privacy [79], the distributed architecture of edge analytics increases the risk of attack vectors [82]. Therefore, revisiting and extending existing privacy and security techniques deserve to be explored further to address the specificity of IoT edge analytics systems. Certainly, these challenges entail not only research and system engineering challenges but also public policy challenges.

Bibliography

[1] J. Lin, W. Yu, X. Yang, P. Zhao, H. Zhang, and W. Zhao, "An edge computing based public vehicle system for smart transportation," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 11, 2020.

[2] S. Amini, I. Gerostathopoulos, and C. Prehofer, "Big data analytics architecture for real-time traffic control," in *2017 5th IEEE international conference on models and technologies for intelligent transportation systems (MT-ITS)*. IEEE, 2017.

[3] Z. Lv, J. Lloret, and H. Song, "Real-time image processing for augmented reality on mobile devices," *Journal of Real-Time Image Processing*, vol. 18, no. 2, 2021.

[4] Hermes. (2019) Real time streaming within augmented and virtual reality. [Online]. Available: https://medium.com/agora-io/sharing-augmented-realities-in-realtime-bfaf9d6f5283

[5] T. BRAUD, P. ZHOU, J. KANGASHARJU, and P. HUI, "Multipath computation offloading for mobile augmented reality," in *2020 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2020.

[6] L.-H. Lee, T. Braud, P. Zhou, L. Wang, D. Xu, Z. Lin, A. Kumar, C. Bermejo, and P. Hui, "All one needs to know about metaverse: A complete survey on technological singularity, virtual ecosystem, and research agenda," *arXiv preprint arXiv:2110.05352*, 2021.

[7] H. Röger and R. Mayer, "A comprehensive survey on parallelization and elasticity in stream processing," *ACM CSUR*, 2019.

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

[9] S. Nayak, R. Patgiri, L. Waikhom, and A. Ahmed, "A review on edge analytics: Issues, challenges, opportunities, promises, future directions, and applications," *arXiv preprint arXiv:2107.06835*, 2021.

[10] S. Li, L. Da Xu, and S. Zhao, "5g internet of things: A survey," *Journal of Industrial Information Integration*, vol. 10, pp. 1–9, 2018.

[11] I. F. Akyildiz, S. Nie, S.-C. Lin, and M. Chandrasekaran, "5g roadmap: 10 key enabling technologies," *Computer Networks*, vol. 106, pp. 17–48, 2016.

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

[13] B. Varghese, E. De Lara, A. Y. Ding, C.-H. Hong, F. Bonomi, S. Dustdar, P. Harvey, P. Hewkin, W. Shi, M. Thiele *et al.*, "Revisiting the arguments for edge computing research," *IEEE Internet Computing*, 2021.

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

[15] J. McChesney *et al.*, "Defog: fog computing benchmarks," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019.

[16] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[17] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms," vol. 52, no. 5, 2019. [Online]. Available: https://doi.org/10.1145/3326066

[18] M. Al-Khafajiy, T. Baker, A. Waraich, D. Al-Jumeily, and A. Hussain, "Iot-fog optimal workload via fog offloading," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. IEEE, 2018.

[19] X. Liu and R. Buyya, "Resource management and scheduling in distributed stream processing systems: a taxonomy, review, and future directions," *ACM CSUR*, 2020.

[20] V. Cardellini, F. Lo Presti, M. Nardelli, and G. Russo Russo, "Run-time adaptation of data stream processing systems: The state of the art," *ACM Computing Surveys (CSUR)*, 2022.

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

[22] V. Cardellini, V. Grassi, F. Lo Presti, and 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.

[23] L. Prosperi, A. Costan, P. Silva, and 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.

[24] M. Nardelli, V. Cardellini, V. Grassi, and F. L. Presti, "Efficient operator placement for distributed data stream processing applications," *IEEE TPDS*, 2019.

[25] A. Tiwari, B. Ramprasad, S. H. Mortazavi, M. Gabel, and E. d. Lara, "Reconfigurable streaming for the mobile edge," in *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, 2019, pp. 153–158.

[26] G. Cookson, "INRIX global traffic scorecard," p. 44, 2018.

[27] W. Hu, Y. Gao *et al.*, "Quantifying the impact of edge computing on mobile applications," in *7th ACM SIGOPS Asia-Pacific Workshop on Systems*, 2016.

[28] S. Katsikeas, K. Fysarakis, A. Miaoudakis *et al.*, "Lightweight & secure industrial iot communications via the mq telemetry transport protocol," in *IEEE SISCC*, 2017.

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

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

[31] M. D. de Assuncao, A. da Silva Veith, and R. Buyya, "Distributed data stream processing and edge computing: A survey on resource elasticity and future directions," *Journal of Network and Computer Applications*, vol. 103, pp. 1–17, 2018.

[32] P. Vít, R. Lukáš, and M. Jan, "loating car data collection for processing and benchmarking," 2018.

[33] H. C. Andrade, B. Gedik, and D. S. Turaga, *Fundamentals of stream processing: application design, systems, and analytics*. Cambridge University Press, 2014.

[34] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[35] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt *et al.*, "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," 2015.

[36] A. Warski, "Windowing data in big data streams-spark," *Flink, Kafka, Akka, Retrieved on May*, vol. 16, p. 2018, 2016.

[37] K. Patroumpas and T. Sellis, "Window specification over data streams," in *International Conference on Extending Database Technology*. Springer, 2006, pp. 445–464.

[38] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," *the VLDB Journal*, vol. 12, no. 2, pp. 120–139, 2003.

[39] B. Gedik, H. G. Özsema, and Ö. Öztürk, "Pipelined fission for stream programs with dynamic selectivity and partitioned state," *Journal of Parallel and Distributed Computing*, vol. 96, pp. 106–120, 2016.

[40] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, C. Soriente, and P. Valduriez, "Streamcloud: An elastic and scalable data streaming system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 12, pp. 2351–2365, 2012.

[41] H. Jafarpour, R. Desai, and D. Guy, "Ksql: Streaming sql engine for apache kafka." in *EDBT*, 2019, pp. 524–533.

[42] R. Tommasini, S. Sakr, E. Della Valle, and H. Jafarpour, "Declarative languages for big streaming data." in *EDBT*, 2020, pp. 643–646.

[43] H. Arkian, G. Pierre, J. Tordsson, and E. Elmroth, "Model-based stream processing auto-scaling in geo-distributed environments," in *ICCCN 2021-30th International Conference on Computer Communications and Networks*, 2021.

[44] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm, "A catalog of stream processing optimizations," vol. 46, no. 4, pp. 1–34. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2597757.2528412

[45] F. Hueske, M. Peters, M. J. Sax, A. Rheinländer, R. Bergmann, A. Krettek, and K. Tzoumas, "Opening the black boxes in data flow optimization," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1256–1267, 2012.

[46] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, "Network-aware operator placement for stream-processing systems," in *22nd International Conference on Data Engineering (ICDE'06)*. IEEE, 2006, pp. 49–49.

[47] S. Rizou, F. Durr, and K. Rothermel, "Solving the multi-operator placement problem in large-scale operator networks," in *2010 Proceedings of 19th International Conference on Computer Communications and Networks*. IEEE, 2010, pp. 1–6.

[48] T. Heinze, L. Aniello, L. Querzoni, and Z. Jerzak, "Cloud-based data stream processing," in *DEBS*, 2014.

[49] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah, "Telegraphcq: continuous dataflow processing," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 2003, pp. 668–668.

[50] R. S. Barga and H. Caituiro-Monge, "Event correlation and pattern detection in cedr," in *International Conference on Extending Database Technology*. Springer, 2006, pp. 919–930.

[51] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina *et al.*, "The design of the borealis stream processing engine." in *Cidr*, vol. 5, no. 2005, 2005, pp. 277–289.

[52] E. A. Rundensteiner, L. Ding, T. Sutherland, Y. Zhu, B. Pielech, and N. Mehta, "Cape: Continuous query engine with heterogeneous-grained adaptivity," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, 2004, pp. 1353–1356.

[53] M. H. Iqbal and T. R. Soomro, "Big data analysis: Apache storm perspective," *International journal of computer trends and technology*, vol. 19, no. 1, pp. 9–14, 2015.

[54] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

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

[56] A. G. Shoro and T. R. Soomro, "Big data analysis: Apache spark perspective," *Global Journal of Computer Science and Technology*, 2015.

[57] H. Arkian, "Resource management for data stream processing in geo-distributed environments," Ph.D. dissertation, Université de Rennes 1, 2021.

[58] B. J. Hansen and T. Bloebaum, "Ffi-notat," 2021.

[59] "Apache nifi," https://nifi.apache.org/, accessed: 2022-03-19.

[60] "Apache edgent," https://www.ibm.com/docs/en/streams/4.2.0?topic=extending-integrating-apache-edgent-streams, accessed: 2022-03-20.

[61] X. Fu, T. Ghaffar, J. C. Davis, and D. Lee, "Edgewise: a better stream processing engine for the edge," in *USENIX ATC*, 2019.

[62] D. Giouroukis, J. Hülsmann, J. von Bleichert, M. Geldenhuys, T. Stullich, F. Gutierrez, J. Traub, K. Beedkar, and V. Markl, "Resense: Transparent record and replay of sensor data in the iot," in *EDBT*, 2019.

[63] A. Kurniawan, *Learning AWS IoT: Effectively manage connected devices on the AWS cloud using services such as AWS Greengrass, AWS button, predictive analytics and machine learning*, 2018.

[64] "Foghorn," https://www.foghorn.io/, accessed: 2022-03-20.

[65] S. Zeuch, A. Chaudhary, B. Del Monte, H. Gavriilidis, D. Giouroukis, P. M. Grulich, S. Breß, J. Traub, and V. Markl, "The nebulastream platform: Data and application management for the internet of things," *arXiv preprint arXiv:1910.07867*, 2019.

[66] P. Mell, T. Grance *et al.*, "The nist definition of cloud computing," 2011.

[67] A. Aggarwal, "Ovhcloud," https://www.ovhcloud.com/en/, 2022.

[68] J. J. J. Geewax, *Google Cloud Platform in Action*. Simon and Schuster, 2018.

[69] B. Golden, *Amazon web services for dummies*. John Wiley & Sons, 2013.

[70] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," pp. 50–55, 2008.

[71] V. Issarny, B. Billet, G. Bouloukakis, D. Florescu, and 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.

[72] P. Varshney and 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.

[73] M. Iorga, L. Feldman, R. Barton, M. Martin, N. Goren, and C. Mahmoudi, "The nist definition of fog computing," National Institute of Standards and Technology, Tech. Rep., 2017.

[74] S. Rizou, F. Diirr, and K. Rothermel, "Fulfilling end-to-end latency constraints in large-scale streaming environments," in *30th IEEE International Performance Computing and Communications Conference*. IEEE, 2011, pp. 1–8.

[75] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy, "Bandwidth estimation: metrics, measurement techniques, and tools," *IEEE network*, vol. 17, no. 6, pp. 27–35, 2003.

[76] S. Khanvilkar, F. Bashir, D. Schonfeld, and A. Khokhar, "Multimedia networks and communication," pp. 431–432, 2004.

[77] B. Lohrmann, P. Janacik, and O. Kao, "Elastic stream processing with latency guarantees," in *2015 IEEE 35th International Conference on Distributed Computing Systems*. IEEE, 2015, pp. 399–410.

[78] H. Röger, S. Bhowmik, and K. Rothermel, "Combining it all: Cost minimal and low-latency stream processing across distributed heterogeneous infrastructures," in *Proceedings of the 20th International Middleware Conference*, 2019, pp. 255–267.

[79] J. Lopez-Fenner, S. Sepulveda, L. F. Bittencourt, F. M. Costa, and N. Georgantas, "Privacy preserving multi party computation for data-analytics in the iot-fog-cloud ecosystem," in *CICCSI 2020: IV International Congress of Computer Sciences and Information Systems*, 2020.

[80] N. Chaurasia, M. Kumar, R. Chaudhry, and O. P. Verma, "Comprehensive survey on energy-aware server consolidation techniques in cloud computing," *JS*, 2021.

[81] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms," *ACM CSUR*, 2019.

[82] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An overview on edge computing research," *IEEE Access*, 2020.

[83] Z. Tao, Q. Xia, Z. Hao, C. Li, L. Ma, S. Yi, and Q. Li, "A survey of virtual machine management in edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1482–1499, 2019.

[84] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, "Extend cloud to edge with kubeedge," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 373–377.

[85] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[86] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[87] C. Puliafito, D. M. Goncalves, M. M. Lopes, L. L. Martins, E. Madeira, E. Mingozzi, O. Rana, and L. F. Bittencourt, "Mobfogsim: Simulation of mobility and migration for fog computing," *Simulation Modelling Practice and Theory*, vol. 101, p. 102062, 2020.

[88] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos, "Enorm: A framework for edge node resource management," *IEEE transactions on services computing*, 2017.

[89] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, and A. Kitazawa, "Fogflow: Easy programming of iot services over cloud and edges for smart cities," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 696–707, 2017.

[90] M. Bauer, E. Kovacs, A. Schülke, N. Ito, C. Criminisi, L.-W. Goix, and M. Valla, "The context api in the oma next generation service interface," in *2010 14th International Conference on Intelligence in Next Generation Networks*. IEEE, 2010, pp. 1–5.

[91] "Openstack," https://www.openstack.org/, accessed: 2022-03-17.

[92] "New: Openinfra foundation edge computing group defines architectures, open source components, and testing activities for massively distributed systems," https://www.openstack.org/use-cases/edge-computing/, accessed: 2022-03-17.

[93] B. C. Şenel, M. Mouchet, J. Cappos, O. Fourmaux, T. Friedman, and R. Mcgeer, "Edgenet: a multi-tenant and multi-provider edge cloud," in *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*, 2021, pp. 49–54.

[94] P. Wang, S. Liu, F. Ye, and X. Chen, "A fog-based architecture and programming model for iot applications in the smart grid," *arXiv preprint arXiv:1804.01239*, 2018.

[95] M. Nardelli, "Qos-aware deployment and adaptation of data stream processing applications in geo-distributed environments," Ph.D. dissertation, Ph. D. thesis, University of Rome Tor Vergata, 2018.

[96] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Optimal operator replication and placement for distributed stream processing systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, no. 4, pp. 11–22, 2017.

[97] S. Kamburugamuve, K. Ramasamy, M. Swany, and G. Fox, "Low latency stream processing: Apache heron with infiniband & intel omni-path," in *Proceedings of the10th International Conference on Utility and Cloud Computing*, 2017, pp. 101–110.

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

[99] U. Srivastava, K. Munagala, and 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.

[100] L. Gu, D. Zeng, S. Guo, Y. Xiang, and J. Hu, "A general communication cost optimization framework for big data stream processing in geo-distributed data centers," *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 19–29, 2015.

[101] W. Chen, I. Paik, and Z. Li, "Cost-aware streaming workflow allocation on geo-distributed data centers," *IEEE Transactions on Computers*, vol. 66, no. 2, pp. 256–271, 2016.

[102] S. Kamburugamuve, L. Christiansen, and G. Fox, "A framework for real time processing of sensor data in the cloud," *Journal of Sensors*, vol. 2015, 2015.

[103] G. Amarasinghe, M. D. De Assuncao, A. Harwood, and 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.

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

[105] F. R. d. Souza, A. D. Silva Veith, M. Dias de Assunção, and 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.

[106] F. R. de Souza, M. D. de Assunçao, E. Caron, and 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.

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

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

[109] M. Peixoto, T. Genez, and L. F. Bittencourt, "Hierarchical scheduling mechanisms in multi-level fog computing," *IEEE Transactions on Services Computing*, 2021.

[110] P. Neophytou, J. Szwedko, M. A. Sharaf, P. K. Chrysanthis, and 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.

[111] P. Neophytou, M. A. Sharaf, P. K. Chrysanthis, and 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.

[112] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 4, pp. 23–32, 2013.

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

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

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

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

[117] T. N. Pham, P. K. Chrysanthis, and A. Labrinidis, "Avoiding class warfare: managing continuous queries with differentiated classes of service," *The VLDB Journal*, vol. 25, no. 2, pp. 197–221, 2016.

[118] M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, and K. Pruhs, "Algorithms and metrics for processing multiple heterogeneous continuous queries," *ACM Transactions on Database Systems (TODS)*, vol. 33, no. 1, pp. 1–44, 2008.

[119] T. Lambert, D. Guyon, and S. Ibrahim, "Rethinking operators placement of stream data application in the edge," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 2101–2104.

[120] F. R. de Souza, M. D. de Assunçao, and 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.

[121] S. Rizou, F. Dürr, and K. Rothermel, "Providing qos guarantees in large-scale operator networks," in *2010 IEEE 12th International Conference on High Performance Computing and Communications (HPCC)*. IEEE, 2010, pp. 337–345.

[122] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.

[123] I. I. C. O. Studio-CPLEX, "Users manual-version 12 release 8," *IBM ILOG CPLEX Division: Incline Village, NV, USA*, 2019.

[124] B. Bixby, "The gurobi optimizer," *Transp. Re-search Part B*, vol. 41, no. 2, pp. 159–178, 2007.

[125] C. Schulte, M. Lagerkvist, and G. Tack, "Gecode," *Software download and online material at the website: http://www. gecode. org*, pp. 11–13, 2006.

[126] J. Forrest and R. Lougee-Heimer, "Cbc user guide," in *Emerging theory, methods, and applications*. INFORMS, 2005, pp. 257–277.

[127] 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.

[128] A. d. Silva Veith, M. D. d. Assunção, and L. Lefevre, "Latency-aware placement of data stream analytics on edge computing," in *International conference on service-oriented computing*. Springer, 2018, pp. 215–229.

[129] G. R. Russo, V. Cardellini, and F. L. Presti, "Reinforcement learning based policies for elastic stream processing on heterogeneous resources," in *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems*, 2019, pp. 31–42.

[130] G. R. Russo, V. Cardellini, G. Casale, and F. L. Presti, "Mead: Model-based vertical auto-scaling for data stream processing," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 314–323.

[131] Y. Wang, Z. Tari, M. R. HoseinyFarahabady, and A. Y. Zomaya, "Model-based scheduling for stream processing systems," in *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2017, pp. 215–222.

[132] T. Z. Fu, J. Ding, R. T. Ma, M. Winslett, Y. Yang, and Z. Zhang, "Drs: Auto-scaling for real-time stream analytics," *IEEE/ACM Transactions on networking*, vol. 25, no. 6, pp. 3338–3352, 2017.

[133] R. K. Kombi, N. Lumineau, and P. Lamarre, "A preventive auto-parallelization approach for elastic stream processing," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 1532–1542.

[134] F. Lombardi, L. Aniello, S. Bonomi, and L. Querzoni, "Elastic symbiotic scaling of operators and resources in stream processing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 3, pp. 572–585, 2017.

[135] H. Zhang, D. Sun, A. Sajjanhar, and R. Buyya, "A data stream prediction strategy for elastic stream computing systems," in *International Conference on Broadband Communications, Networks and Systems*. Springer, 2021, pp. 148–162.

[136] Z. Hu, H. Kang, and M. Zheng, "Stream data load prediction for resource scaling using online support vector regression," *Algorithms*, vol. 12, no. 2, p. 37, 2019.

[137] T. Heinze, Z. Jerzak, G. Hackenbroich, and C. Fetzer, "Latency-aware elastic scaling for distributed data stream processing systems," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, 2014, pp. 13–22.

[138] V. Cardellini, F. L. Presti, M. Nardelli, and G. R. Russo, "Decentralized self-adaptation for elastic data stream processing," *Future Generation Computer Systems*, vol. 87, pp. 171–185, 2018.

[139] V. Kalavri, J. Liagouris, M. Hoffmann, D. Dimitrova, M. Forshaw, and T. Roscoe, "Three steps is all you need: fast, accurate, automatic scaling decisions for distributed streaming dataflows," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 783–798.

[140] B. Lohrmann, D. Warneke, and O. Kao, "Massively-parallel stream processing under qos constraints with nephele," in *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, 2012, pp. 271–282.

[141] C. Balkesen, N. Tatbul, and M. T. Özsu, "Adaptive input admission and management for parallel stream processing," in *Proceedings of the 7th ACM international conference on Distributed event-based systems*, 2013, pp. 15–26.

[142] F. R. De Souza, "Scheduling solutions for data stream processing applications on cloud-edge infrastructure," Ph.D. dissertation, Université de Lyon, 2020.

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

[144] T. Djemai, P. Stolf, T. Monteil, and J.-M. Pierson, "Mobility support for energy and qos aware iot services placement in the fog," in *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE, 2020, pp. 1–7.

[145] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *JACM*, 1972.

[146] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 15–26, 2004.

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

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

[149] B. Jansson, "Choosing a good appointment system—a study of queues of the type (d, m, 1)," *Operations Research*, vol. 14, no. 2, pp. 292–312, 1966.

[150] P. Ntumba, N. Georgantas, and 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.

[151] ——, "Efficient scheduling of streaming operators for iot edge analytics," in *Accepted to the 6th International Conference on Fog and Mobile Edge Computing*, 2021.

[152] H. M. Salkin and C. A. De Kluyver, "The knapsack problem: a survey," *Naval Research Logistics Quarterly*, 1975.

[153] C. H. Papadimitriou, "On the complexity of integer programming," *J. ACM*, vol. 28, no. 4, p. 765–768, oct 1981. [Online]. Available: https://doi.org/10.1145/322276.322287

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

[155] V. A. Strusevich, "Shop scheduling problems under precedence constraints," *Annals of operations research*, vol. 69, pp. 351–377, 1997.

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

[157] E. Balas, G. Lancia, P. Serafini, and A. Vazacopoulos, "Job shop scheduling with deadlines," *Journal of Combinatorial Optimization*, vol. 1, no. 4, pp. 329–353, 1998.

[158] J. Stastny, V. Skorpil, Z. Balogh, and R. Klein, "Job shop scheduling problem optimization by means of graph-based algorithm," *Applied Sciences*, vol. 11, no. 4, p. 1921, 2021.

[159] H. Suleiman and O. Basir, "Qos-driven job scheduling: Multi-tier dependency considerations," *arXiv preprint arXiv:2004.05695*, 2020.

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

[161] C. Y. Zhang, P. Li, Y. Rao, and Z. Guan, "A very fast ts/sa algorithm for the job shop scheduling problem," *Computers & Operations Research*, vol. 35, no. 1, pp. 282–294, 2008.

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

[163] P. J. Van Laarhoven, E. H. Aarts, and J. K. Lenstra, "Job shop scheduling by simulated annealing," *Operations research*, vol. 40, no. 1, pp. 113–125, 1992.

[164] H. Hosseini Nasab and F. Mobasheri, "A simulated annealing heuristic for the facility location problem," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 3, pp. 210–224, 2013.

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

[166] C. Wöbker, A. Seitz, H. Mueller, and B. Bruegge, "Fogernetes: Deployment and management of fog computing applications," in *Network Operations and Management Symposium*. IEEE, 2018, pp. 1–7.

[167] "7-Zip LZMA Benchmark." [Online]. Available: https://www.7-cpu.com/

[168] U. Tadakamalla and D. A. Menascé, "Characterization of iot workloads," in *International Conference on Edge Computing*. Springer, 2019, pp. 1–15.

[169] D. O'Keeffe, T. Salonidis, and P. Pietzuch, "Frontier: Resilient edge processing for the internet of things," *Proceedings of the VLDB Endowment*, vol. 11, no. 10, pp. 1178–1191, 2018.

[170] M. Chao and R. Stoleru, "R-mstorm: A resilient mobile stream processing system for dynamic edge networks," in *2020 IEEE International Conference on Fog Computing (ICFC)*. IEEE, 2020, pp. 64–72.

Appendix A

Use case detailed operators

In the following we detail the operators that constitute the DSPA application introduced in the use example for country wide traffic monitoring in Section 1.1:

Union (U): combines raw traffic data streams from street antennas into a single data stream for further processing:

```
CREATE STREAM unionDataStream as
UNION SELECT car_id, latitude, longitude, speed, timestamps
FROM S_1,S_2,...,S_N
WINDOW HOPPING (SIZE 1 SECOND, ADVANCED BY 1 SECOND);
```

Join ($\bowtie^R$): matches the GPS-location of vehicles and the road network information (R) to find-out the street segment and the city related to the GPS-location of the vehicle:

```
CREATE STREAM joinDataStream as
SELECT d.car_id, d.speed, r.segment, r.city
FROM unionDataStream as d, raodNetwork as r
WHERE d.latitude = r.latitude and d.longitude = r.longitude
WINDOW HOPPING (SIZE 1 SECOND, ADVANCED BY 1 SECOND );
```

Group (G_{by}): groups the input data stream per street segment, computes the average speed and the number of vehicles per street segment:

```
CREATE STREAM aggregationDataStream as
SELECT d.segment, d.city, count(d.car_id) as car_nbr, avg(d.speed) as avg_speed
FROM joinDataStream as d
GROUP BY d.segment
WINDOW HOPPING (SIZE 1 SECONDS, ADVANCED BY 1 SECONDS );
```

Split (SP): copies the input data stream in two data streams, e.g., the first for supporting country-wide traffic monitoring and the second city-wide traffic regulation:

```
CREATE STREAM countryWideDataStream as
SELECT d.city, d.segment, d.car_nbr, d.avg_speed
CREATE STREAM cityWideDataStream as
```

```
SELECT d.city,d.segment, d.car_nbr, d.avg_speed
FROM aggregationDataStream as d
WINDOW HOPPING (SIZE 1 SECONDS, ADVANCED BY 1 SECONDS );
```

The resulting data stream after applying the previous operators is given as input for the country-wide traffic monitoring and city-wide traffic regulation IoT applications.

The first application for country-wide traffic monitoring reports on an hourly basis traffic statistics for the entire country. In this respect, we need to group the input data stream per city to provide the average speed and number of vehicles per city:

```
CREATE STREAM aggregationDataStream as
SELECT d.city, sum(d.car_nbr) as car_nbr_city, avg(d.avg_speed) as avg_speed
FROM countryWideDataStream as d
GROUP BY d.city
WINDOW HOPPING (SIZE 3600 SECONDS, ADVANCED BY 3600 SECONDS );
```

and finally filter (σ) the result of the previous operator to obtain the global traffic status per city. We assume that the first application allows a geographical browsing by city, hence the operator σ is executed several times with different constant values captured by the parameter \$CITY_ID:

```
CREATE STREAM results as SELECT d.city, d.car_nbr_city, avg_speed
FROM aggregationDataStream as d
WHERE d.city = \$CITY_ID
WINDOW HOPPING (SIZE 3600 SECONDS, ADVANCED BY 3600 SECONDS );
```

The second application for city-wide traffic regulation aims to support the control of the traffic lights e.g., to give priority to jammed traffic flows over non-jammed ones, etc. In this respect, we need to simply filter (σ) the input data stream for the specified city and obtain the traffic per street segment:

```
CREATE STREAM results as
SELECT d.segment, d.avg_speed, d.car_nbr
FROM cityWideDataStream as d
WHERE d.city = \$CITY_ID
WINDOW HOPPING (SIZE 20 SECONDS, ADVANCED BY 20 SECONDS);
```

Appendix B

Data stream rate sequences

Sequence of random data stream rates in Chapter 5

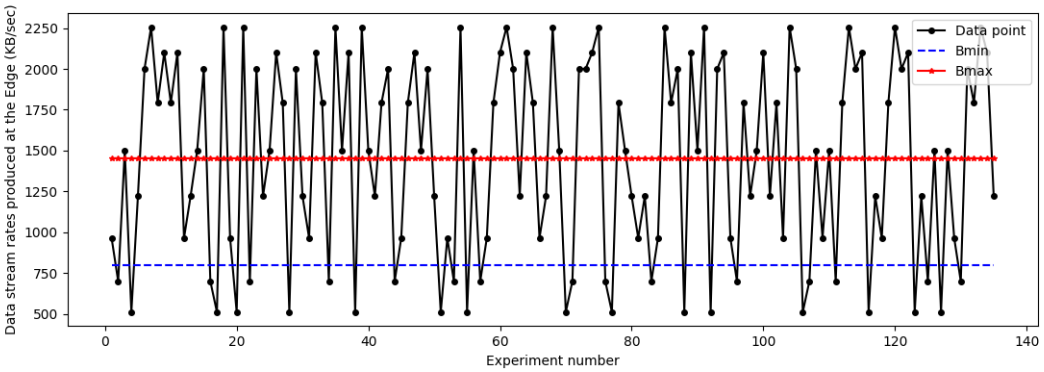


Figure B.1: Simulation of 9 data point evolving randomly

Sequence of random data stream rates in Chapter 6

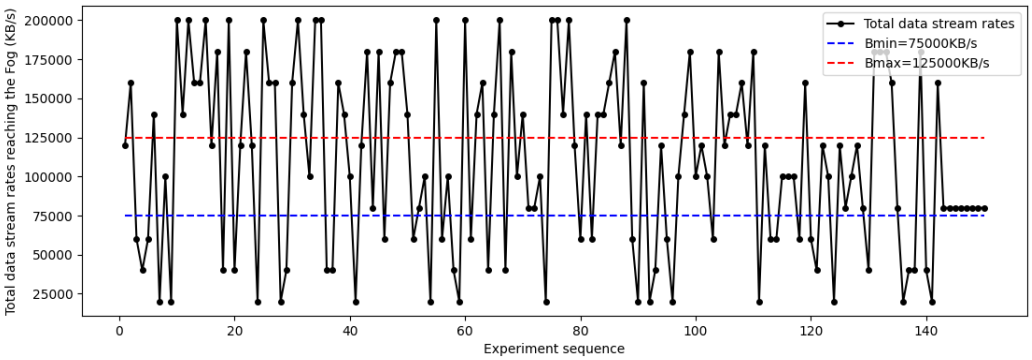


Figure B.2: Simulation of 10 data points evolving randomly used in Chapter 6

Sequence of random data stream rates in Chapter 7

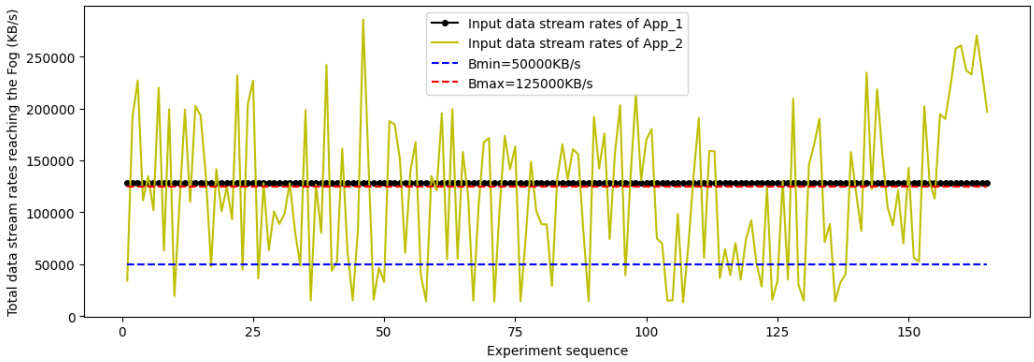


Figure B.3: Simulation of 11 data points evolving randomly used in Chapter 7 for scenario 1

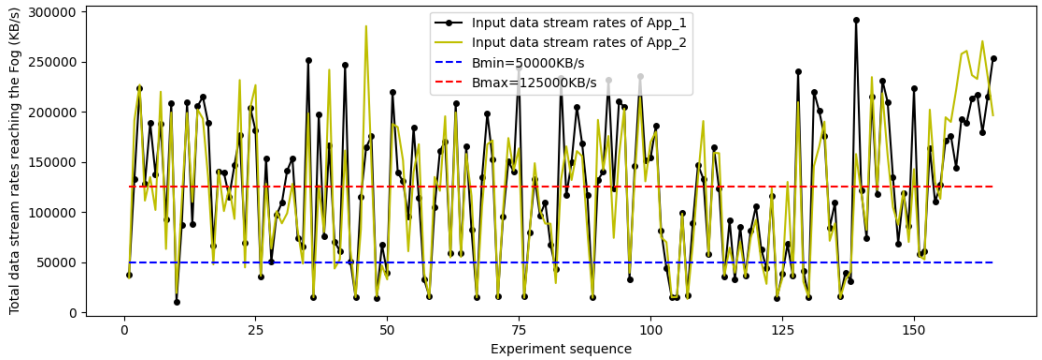


Figure B.4: Simulation of 11 data points evolving randomly used in Chapter 7 for scenario 2

Appendix C

List of publications

In conference proceedings

- **Patient Ntumba**, Nikolaos Georgantas, and Vassilis Christophides, *Scheduling of Continuous Operators for IoT edge Analytics with Time Constraints* In the International Conference on Smart Computing, Jun 2022, Espoo, Finland.

  `https://hal.inria.fr/hal-03413549`.

- **Patient Ntumba**, Nikolaos Georgantas, and Vassilis Christophides, *Efficient Scheduling of Streaming Operators for IoT Edge Analytics* In the 6th International Conference on Fog and Mobile Edge Computing, 2021, Gandia, Spain.

  `https://hal.inria.fr/hal-03413549`.

- **Patient Ntumba**, Nikolaos Georgantas, and Vassilis Christophides, *Scheduling Continuous Operators for IoT Edge Analytics.* In the 4th International Workshop on Edge Systems, Analytics and Networking colocated with EuroSys'21, Apr 2021, Online United Kingdom, United Kingdom.

  `https://hal.inria.fr/hal-03208518v2`.

In Journal proceedings

- Georgios Bouloukakis, Nikolaos Georgantas, **Patient Ntumba**, Valérie Issarny, *Automated synthesis of mediators for middleware-layer protocol interoperability in the IoT* In the Future Generation Computer Systems, Elsevier, 2019. `https://hal.inria.fr/hal-02304074`.

Demos, posters

- **Patient Ntumba**, Nikolaos Georgantas, and Vassilis Christophides, *PhD earlier stage: IoT data stream analytics in the edge-fog-cloud continuum* Poster at JOURNEE EDITE, September 2019, Paris, France.
- **Patient Ntumba**, Georgios Bouloukakis, Nikolaos Georgantas, *Interconnecting and Monitoring Heterogeneous Things in IoT Applications* Demo paper in International Conference on Web Engineering, Jun 2018, Caceres, Spain. `https://hal.inria.fr/hal-01771857`.

List of figures

List of tables