Scheduling Solutions for Data Stream Processing Applications on Cloud-Edge Infrastructure

Felipe Rodrigo de Souza

Numéro National de Thèse : 2020LYSEN082

THÈSE de DOCTORAT DE L'UNIVERSITE DE LYON

opérée par

l'École Normale Supérieure de Lyon

École Doctorale N°512
Informatique et Mathématiques de Lyon

Discipline : Informatique

présentée et soutenue publiquement le 10/12/2020, par :

Felipe Rodrigo DE SOUZA

Scheduling Solutions for Data Stream Processing Applications on Cloud-Edge Infrastructure

Solutions de planification pour les applications de traitement de flux de données sur une infrastructure Cloud-Edge

Devant le jury composé de :

Gabriel ANTONIU	Directeur de Recherche Inria	*Rapporteur*
Rajiv RANJAN	Professeur à l'Université de Newcastle	*Rapporteur*
Frédéric LE MOUËL	Professeur à l'Université de Lyon	*Examinateur*
Sébastien MONNET	Professeur à l'Université Savoie Mont Blanc	*Examinateur*
Patricia STOLF	Maître de Conférences à l'Université Paul Sabatier	*Examinatrice*
Valeria CARDELLINI	Professeure Associée à l'Univ. de Rome Tor Vergata	*Examinatrice*
Eddy CARON	Maître de Conférences HDR ENS de Lyon	*Directeur*
Marcos DIAS DE ASSUNÇÃO	Docteur	*Co-encadrant*

Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my deepest gratitude to my advisor Dr. Marcos Dias de Assunção, for the many discussions, the continuous support, and motivation to reach the milestone of writing this thesis. I also would like to thank Eddy Caron for the incentives and enthusiastic participation during this process. Both of them helped me to achieve this goal and to whom I will always grateful.

I would like to thank my family and close friends. Whom stood by my side through this journey, giving me support and strength to pursue this goal. A special thanks to my father and mother, who always support no matter my decision and always guided me to make smart decisions. There are no words to express my gratitude.

Finally, I would like to thank my pears at the AVALON team, welcoming me and providing tools to achieve this thesis. A distinguished thank you to the former AVALON member and office mate Alexandre Da Silva Veith, with I had many discussions and collaborations that helped me improve my work.

Abstract

Technology has evolved to a point where applications and devices are highly connected and produce ever-increasing amounts of data used by organizations and individuals to make daily decisions. For the collected data to become information that can be used in decision making, it requires processing. The speed at which information is extracted from data generated by a monitored system or environment affects how fast organizations and individuals can react to changes. One way to process the data under short delays is through Data Stream Processing (DSP) applications. DSP applications can be structured as directed graphs, where the vertexes are data sources, operators, and data sinks, and the edges are streams of data that flow throughout the graph. A data source is an application component responsible for data ingestion. Operators receive a data stream, apply some transformation or user-defined function over the data stream and produce a new output stream, until the latter reaches a data sink, where the data is stored, visualized or provided to another application.

Usually, DSP applications are designed to run on cloud computing or on a homogeneous cluster of computing resources, due to the large set of resources that such infrastructures can provide and the high speed network used to interconnect the resources. In scenarios where the data consumed by the DSP application is produced on the cloud itself, deploying the entire application on the cloud looks like a sensible approach. However, as the Internet of Things (IoT) becomes more pervasive, there is an increasing number of scenarios where DSP applications consume data streams that are generated at the edges of the network, by numerous geographically distributed devices. In such scenarios, sending all the data through the Internet to be processed on a distant cloud, far from the network edges where the data was originated, leads to considerable network traffic, hence substantially increasing the application end-to-end latency; the time from when the data is collected until processing completes.

Edge computing has emerged as a paradigm to offload processing tasks from the cloud to resources located more closely to data sources. Edge computing resources, however, are often more constrained than those available in the cloud. A recent trend consists in exploring the combination of cloud and edge computing resources to execute DSP applications. Although the combined use of cloud and edge computing resources is sometimes referred to as *fog computing*, the research community does not seem to have reached a consensus on the terminology. We call the combination of cloud and edge computing resources as *cloud-edge* infrastructure.

Regardless the terminology in place, the rationale is that resources located at the edges of the network can provide low latency, process parts of the application, and reduce the amount of data sent to the cloud, whereas the cloud can be used as a central place that receives data from geographically distributed streams. The drawback of using resources at the network edges, however, is that these resources are constrained with respect to CPU, memory, storage, and even power availability. When scheduling applications on cloud-edge infrastructure, in addition to solving the operator placement problem, which consists of finding a set of resources to host

the operators of a DSP application, a scheduling solution needs to consider the computing constraints of edge resources. It must decide which parts of the application to offload to the edge and which parts keep in the cloud. Moreover, the solution must explore the resources at the edges of the network by adapting the application parallelism to split the application load among the numerous edge resources to cope with the application load. An operator whose requirements can be easily met by a single cloud resource may have to be transformed into multiple replicas – each of which process less data – deployed across a number of edge computing resources. This gives rise to two inter-related problems, namely the operator placement and the degree of parallelism of each operator.

In this thesis, we propose a model for the operator placement and parallelism problems, accounting for both cloud and edge computing resources and their heterogeneity. The model addresses the operator parallelism by creating multiple replicas to explore a large number of computationally constrained devices at the network edges. Along with the model, we propose an optimal solution based on linear formulation to reduce the application end-to-end latency and deployment costs. Since an optimal solution based on linear formulation suffers from scalability issues – and a cloud-edge infrastructure may comprise a large number of resources – we propose a heuristic-based approach to reduce the search space without compromising the solution's performance. Simulation results show that the proposed solution can achieve an application end-to-end latency at least $\simeq 80\%$ and monetary costs at least $\simeq 30\%$ better than a traditional cloud deployment that places the entire application on the cloud. When combining the proposed solution with the heuristic approach to reduce the search space, it can find placements 94% faster, with a performance degradation of 12%, which is still much better than state-of-the-art approaches.

French Abstract

L'évolution des technologies ont conduit à une forte connection entre les applications et le matériel produisant des quantités de données en perpétuelle augmentation. Ces données sont utilisées par les entreprises, les organisations et les individus pour prendre des décisions quotidiennes. Pour que les données collectées soient réellement utiles il convient de les traiter à temps et donc suffisament rapidement. La vitesse à laquelle les informations sont extraites depuis les données générées par un système ou un environnement surveillé a un impact sur la capacité des entités (entreprises, organisations ou individus) à réagir aux changements. Une solution pour le traitement des données dans un délais réduit consiste à utiliser des applications de traitement de flux de données. Les applications de traitement de flux de données peuvent être modélisées sous forme de graphes orientés, où les sommets sont des sources de données, des opérateurs ou des récepteurs de données (*i.e., data sinks*), et les arêtes représentent les flux de données entre les opérateurs. Une source de données est un composant d'application responsable de la génération des données. Les opérateurs reçoivent un flux de données, appliquent une transformation ou effectuent une fonction définie par l'utilisateur sur le flux de données entrant et produisent un nouveau flux de sortie, jusqu'à ce que ce dernier atteigne un récepteur de données, où les données sont alors stockées, visualisées ou envoyées à une autre application.

Habituellement, les applications de traitement de flux de données sont conçues pour fonctionner sur des infrastructures *cloud* ou sur une grappe homogène de ressources (*i.e., cluster*) en raison du nombre de ressources que ces infrastructures peuvent fournir et de la bonne connectivité de leur réseau. Dans les scénarios où les données utilisées par l'application de traitement du flux de données sont produites dans le cloud lui-même alors le déploiement de l'ensemble de l'application sur le cloud est une approche pertinente. Cependant, à l'heure où l'Internet des objets devient de plus en plus omniprésent, il existe un nombre croissant de scénarios où les applications de traitement de flux de données consomment des flux de données générés à la périphérie du réseau (via les nombreux appareils et capteurs répartis géographiquement). Dans de tels scénarios, l'envoi de toutes les données via Internet pour être traitées sur un cloud distant, loin de la périphérie du réseau d'où proviennent les données, conduirait à générer un trafic réseau considérable. Cela augmente ainsi de façon significative la latence de bout en bout pour l'application; c'est-à-dire, le délai entre le moment où les données sont collectées et la fin du traitement.

L'informatique de périphérie (*edge computing*) est devenu un paradigme pour aléger les tâches de traitement du cloud vers des ressources situées plus près des sources de données. Cependant, les ressources informatiques de périphérie sont souvent plus limitées en puissance et en capacité que celles disponibles dans le cloud. Une tendance récente consiste à explorer la combinaison de ressources cloud et périphérique pour exécuter des applications de traitement de flux de données. Bien que l'utilisation combinée de ces ressources soit parfois appelée *fog computing*, la communauté scientifique ne semble pas avoir atteint un consensus sur la terminologie. Nous

appelons la combinaison de ressources cloud et de ressources périphériques une infrastructure *cloud-edge*.

Quelle que soit la terminologie choisie, il est classique de constater que les ressources situées en périphérie du réseau offre une faible latence, et ne peuvent traiter qu'une partie des données de l'application mais permettent cependant de réduire la quantité de données envoyées au cloud. Le cloud peut être utilisé comme un lieu pour centraliser les flux de données distribués géographiquement. Cependant, l'inconvénient de l'utilisation de ressources périphériques, est qu'elles sont limitées d'un point de vue de leur puissance processeur, de leur mémoire, de leur capaité de stockage et même de la disponibilité de l'énergie pour les alimenter. Lors de la planification et l'ordonnancement d'applications sur une infrastructure cloud-edge, en plus de résoudre le problème de placement des opérateurs, qui consiste à trouver un ensemble de ressources pour héberger ces opérateurs, une solution de planification doit aussi prendre en compte les contraintes de calcul des ressources périphériques. Cette solution doit déterminer quelles parties de l'application est à placer vers la périphérie de l'infrastructure et quelles parties restent à la charge du cloud. De plus, la solution doit analyser les ressources en périphérie du réseau en adaptant le parallélisme applicatif pour répartir la charge applicative entre les nombreuses ressources pour faire face à la charge applicative. Un opérateur dont les besoins peuvent être facilement satisfaits par une seule ressource cloud peut être transformé en plusieurs instances - chacune traitant moins de données - déployées sur un certain nombre de ressources périphériques. Cela pose deux problèmes concomitants, à savoir le placement des opérateurs et l'optimisation du degré de parallélisme de chaque opérateur.

Dans cette thèse, nous proposons un modèle qui adresse les problèmes de placement et de parallélisme des opérateurs, tenant compte à la fois des ressources de cloud et de ressources périphériques ainsi que leurs hétérogénéités. Le modèle aborde le parallélisme des opérateurs en créant plusieurs instances pour explorer un grand nombre de ressources périphériques. Parallèlement au modèle, nous proposons une solution optimale basée sur une formulation linéaire pour réduire la latence de bout en bout ainsi que les coûts de déploiement. Étant donné qu'une solution optimale basée sur une formulation linéaire souffre de problèmes d'évolutivité - et qu'une infrastructure en périphérie du cloud peut inclure un grand nombre de ressources - nous proposons une approche heuristique pour réduire l'espace de recherche sans sacrifier les performances de la solution. Les résultats des simulations montrent que la solution proposée peut atteindre une latence de bout en bout d'environ 80% de moins et des coûts monétaires diminué d'environ 30% par rapport à un déploiement cloud traditionnel qui centralise l'ensemble de l'application. En combinant la solution proposée avec l'approche heuristique pour réduire l'espace de recherche, la solution proposée peut accélérer de 94% les déploiements, avec une dégradation des performances de la plateforme de seulement 12% (ce qui reste acceptable).

Chapter 1

Introduction

Contents

Over the past year, computing has become very pervasive to most areas and environments of society, generating ever increasing amounts of data [1]. These large amounts of data are usually used to extract valuable information in a process commonly referred as *big data* [2]. With a lot of information gathered by connected systems and big data, our society emerged into a *data driven economy*, where individuals and organizations would take decisions over collected data [3]. This benefits areas going from fraud detection, resource management, health care and natural disaster management [4, 5]. Extracting valuable information from large volumes of data requires processing, often following mostly two common approaches:

- Batch Processing: An approach that applies a store-and-process scheme, where a large volume of data is stored in data centers, and then processed using programming models such as MapReduce [5].
- Data Stream Processing (DSP): This approach applies on-the-fly processing, where the data is processed as it is generated, hence producing near real-time results [5].

Batch processing is widely adopted by individuals and organizations[6], but according with IBM in 2012, 2.5 exabytes of data were generated daily [7]. On the data driven economy the speed in which individuals and organizations take actions is critical. Then, processing exabytes of data daily almost in real time to extract valuable information, using a store-and process approach became unfeasible [8, 9]. With the demand for fast processing DSP applications posed as a solution. In scenarios such as stock exchange, fraud detection, health care, and many others, fast decisions are crucial [10].

Usually DSP applications are designed as *dataflow* graphs, as depicted in Figure 1.1, with vertexes representing operators, data sources, and data sinks and edges representing how data

flows between operators [11]. Data sources are the part of the application responsible for data ingestion, whereas data sinks consume the processed data, providing it to end-users, to other systems, or acting on it. DSP applications have at least one source and one sink. Between sources and sinks, each data message traverses a series of operators, where the data is transformed, discarded, or replicated. An operator is a processing unit that receives a continuous incoming stream, applies any user-defined function on it, and generates a new output stream [12]. Some operators require maintaining a state for further processing, for instance, operators that compute average values, operators that discard messages according to some criteria. Operators can be categorized according to their state, where an operator can be *stateless*, in which case it does not maintain any state between executions; *partitioned stateful* where a given data structure maintains state for each down stream based on a partitioning key, and *stateful* where no particular structure is required [13].

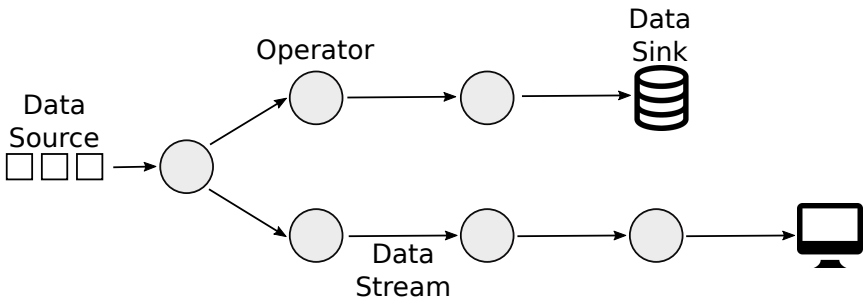


Figure 1.1: Data stream processing application.

Commonly, DSP applications are deployed on clouds to leverage the virtually unlimited number of resources that clouds can provide [14]. The cloud deployment approach allows DSP applications to achieve high scalability and handle large data streams. Cloud deployment is a suitable solution for DSP applications that either do not have very stringent latency requirements or that process data originated in the cloud itself. An example is web analytics, where the web application is hosted in the same cloud and provides data to the DSP application which aims to identify patterns, security breaches, etc. [15].

The Internet of Things (IoT) industry has experienced a huge growth over the past few years, likely reaching around 15 millions of devices in 2019 [16]. According to Cisco, it is expected that by 2025 there will be more than 75 million IoT devices [17]. This scenario places a large number of devices at the edges of the network, generating ever-increasing data streams that need to be analyzed under short delays and from which valuable information must be extracted. Forwarding these data streams from the edges of the Internet to a distant cloud for processing generates a lot of network traffic and incurs high application end-to-end latency, which is beyond the near real-time requirements of DSP applications [11, 18].

DSP applications applied to process data collected from IoTs or sensors, can be found in the aviation, where DSP applications are used to evaluated data collected from sensors spread through the airplane, to make sure that the collected data is reliable and that the pilot can make decisions over the information obtained from the data [19]. Another example is traffic control, where multiple cameras as sensors are spread through the city continuously generating data that is sent to a cloud for processing and then decisions over open freeways, traffic lights timer are taken to reduce traffic congestion [20]. Energy grid management with IoT resources spread through the grid generating data that will be collected and analyzed to identify anomalies, frauds, blackouts etc. [21], among many other scenarios.

To improve the performance of DSP in IoT scenarios a recent trend consists in exploring, in addition to cloud resources, resources that are at the edges of the network [5, 21, 22]. Existing work sometimes calls the combination of cloud and edge computing resources as *fog computing*, although the use of this terminology is not unanimous. In this thesis, we refer to the combination of cloud and edge computing resources produces as cloud-edge infrastructure, as depicted in Figure 1.2.

The considered cloud-edge infrastructure is three-layered [23, 24]. The first layer, namely IoT layer, contains numerous geo-distributed resources commonly explored just as data sources, data sinks, or actuators. Resources on the IoT layer are sensors and IoT devices, and these devices are grouped under different domains, composing a site. IoT resources have several constraints in terms of CPU, memory, storage, and even power source, but the computing capabilities of such devices are non-negligible and can be explored. The second layer, named *MD*, contains limited sets of resources, geographically distributed but with low latency to IoT resources. Resources on this layer are routers, gateways, and Micro Datacenter (MD). Regarding computational power, MD resources are still constrained but more capable than resources in the IoT layer. IoT and *MD* layers are considered here as edge computing resources. The third layer, the cloud, contains all the powerful and virtually unlimited resources that any cloud can provide at a high latency distance from the application data sources.

Due to the computing constraints of edge resources, operators offloaded from the cloud usually do not have their requirements met by a single resource [25]. Then, despite the low network latency provided by edge resources, DSP applications' overall performance might face degradation. A common approach to circumvent this problem aims to explore *operator parallelism*, where multiple replicas or instances of each operator are deployed [26]. Each replica is responsible for processing a fraction of the overall operator's load, hence requiring less computational power. Since edge resources are numerous, using distributed processing, with replicas deployed on different resources, they handle the overall load of the operator.

There is a lack of consensus regarding how edge resources are provided for deployment, at what costs, and what are their computational capacities. The IoT part of the edge infrastructure usually contains application-focused resources, making it difficult for an Infrastructure Service Provider (ISP) to offer resources that both meet the specific demand of DSP applications and are generic enough to support many applications. Since MD resources are not application focused, it is possible to find some deployment options on major Infrastructure as a Service (IaaS) providers as Amazon AWS Local Zones[1] or AWS Outpost[2], and other small providers. This work considers a cloud-edge infrastructure as a combination of public and private resources, where IoT resources are considered private and MD and cloud resources are public.

The process of deciding on which resources to deploy the operators of a DSP application, commonly know as *operator placement problem* [15, 27], aims to find a set of resources with computational capacities greater than or equal to the sum of requirements of all operators. The operator placement problem is guided through the optimization of performance metrics such as application end-to-end latency and throughput, or other Quality of Service (QoS) metrics (*i.e.*, monetary cost). The search for a solution to the operator placement problem is known to be NP-Hard [28]. The cloud-edge infrastructure poses additional challenges, due to the wide range of heterogeneous devices, computational constraints, and even the network between each layer, that usually crosses the Internet and faces traffic congestion.

[1] https://aws.amazon.com/about-aws/global-infrastructure/localzones/
[2] https://aws.amazon.com/outposts/

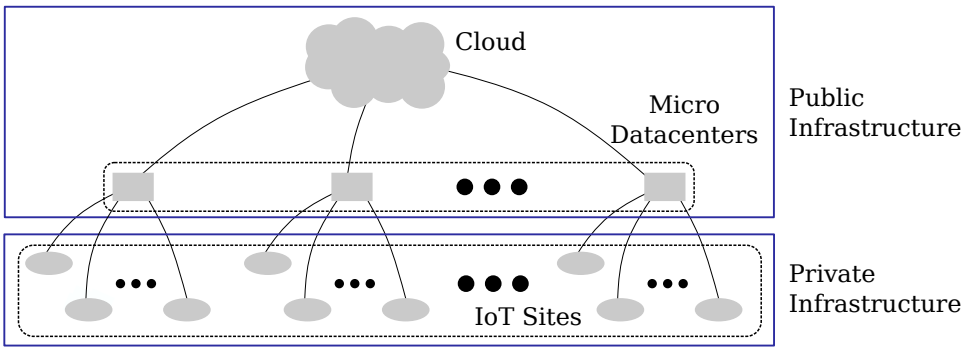


Figure 1.2: Cloud-edge infrastructure overview.

1.1 Challenges in Operator Placement for DSP Applications

The performance metrics that guide a scheduling solution are directly affected by operators' requirements, given in terms of CPU, memory, and storage. The ability of a scheduling solution to provide a placement that guarantees performance improvement depends on how it models the performance metrics and the operator's placement. Even if the performance metrics and requirements are adequately modeled, if the requirements are not met, the scheduling solution cannot perform a suitable placement. These requirements are often specified by application owners with little to no knowledge to do so.

Another challenging aspect of the operator placement problem addressing the cloud-edge infrastructure is the large resource heterogeneity and geographical distribution. DSP applications are time-sensitive, with stringent time constraints, with which the cloud can deal to some extent by providing more powerful resources, but there are still physical limitations due to its distance from the data sources. The alternative is the utilization of edge resources with lower latency to the data sources but with lower computational power. To complicate matters further, deployments on public infrastructure such as MDs and cloud incur a monetary cost, that should be considered.

1.1.1 Edge Computing

Edge computing pushes computation to the edges of the network, through several highly distributed resources. It is a paradigm that can be used to leverage the performance of mobile computing [29], or any application that receives data generated at the edges of the network, such as DSP applications [30]. There is, however, a lack of consensus regarding the definition of edge computing. Existing work considers edge computing as numerous IoT devices collecting data at the edges of the network [21, 31, 32], other works consider MD, routers etc. [14, 33, 34], and part of the existing work, like in this thesis, consider edge computing as a two-layered infrastructure with one layer containing IoT devices, and another layer with MD, routers and gateways [35, 36, 37].

While the edge computing approach allows for new possibilities regarding near real-time for IoT applications, it lacks computational power. Some edge computing resources are gateways, routers, and small computers often with a fraction of the computational power of cloud resources. To give a little more support to edge computing, MDs are able to provide more powerful resources, but still far from their cloud counterparts. The limited computational capacity of edge computing resources commonly does not meet the requirements of operators offloaded

from the cloud. Thus, without any technique such as operator replication, to reduce the requirements of each operator's instance, it would be counter productive to deploy operators on edge resources that cannot meet the operator's requirements, resulting in overall performance degradation. Hence, the goal is to identify which operators can be offloaded from the cloud, that would benefit from low network latency provided from edge computing, and how to offload such operators in terms of requirements.

1.1.2 Operator Placement and Replication

To properly explore edge computing resources, scheduling solutions need to determine the most suitable operators for edge resources. Beyond this decision, it needs to adjust the requirements used to configure each operator, due to computational constraints from edge resources. Even after deciding to use edge computing, it needs to decide between IoT or MD resources since they face different limitations and offer different benefits.

Because edge computing resources have computational constraints, especially IoT ones, and DSP applications require timely processing of large volumes of data, resource with such computational constraints might not pose as the most suitable ones. Scheduling solutions circumvent computational constraints of edge computing through the creation of multiple replicas per operator with significantly smaller computational requirements and balancing the load between such replicas. Then, scheduling solutions need to compute the number of replicas, where to place them and the computational requirements given to each replica.

1.2 Research Problems and Objectives

My research focuses on the placement of DSP applications with operator replication and bandwidth guarantees on cloud-edge infrastructure to improve performance and QoS metrics. To tackle this challenge, we seek to meet the following objectives:

- **How to model a DSP application on cloud-edge infrastructure?** Commonly mathematical models are created based on a linear formulation. Some of these models include Queueing Theory elements; others consider just integers; other models mix integers with non-integer numbers. Currently, models for DSP applications are usually oversimplified. Some create a very simplistic model of operators, or ignore the data streams between them. The infrastructure often faces the same oversimplifications, especially regarding the network. With the inclusion of edge resources and their limitations, aspects such as different clock speeds, CPU, and memory capacities are neglected.

- **How to create replicas and distribute the load of each operator?** To avoid overload computationally constrained resources at the edge, operator replication becomes a requirement. Even when accounting only for cloud deployment, scheduling solutions already explore operator replication. Most models base their operator replication on a maximum number of replicas provided by the user, which again can lead to overloaded resources. The challenge relies on deciding on the number of replicas, how to distribute the load to each replica, and the resource capacity to provide to each replica

- **How to place DSP applications considering a multi-objective optimization?** Most DSP frameworks use as a scheduling solution round-robin techniques, which essentially does not provide any performance optimization. Even scheduling solutions optimize

performance metrics optimize one at a time, and more common than not, QoS metrics are neglected.

1.3 Evaluation Methodology

We provide a mechanism for placing DSP applications across cloud-edge infrastructure, with operator replication, bandwidth guarantees, in a multi-objective optimization. The mechanisms proposed in this document are evaluated using real environment and discrete-event simulations in order to create controllable environments for repeatable experiments. While for experiments performed in real environment we used a sentiment analysis application [38], the simulated applications are crafted based on the shape and load of state-of-the-art evaluations and on a benchmark for real-time IoT applications, RIoTBench [39]. Applications have variable communication patterns and operators' requirements.

1.4 Contributions

The **key contributions** of this thesis are listed below:

1. An optimal mathematical model based on Mixed Integer Linear Programming (MILP) for DSP operator placement and replication with optimization of end-to-end latency and deployments costs, and throughput guarantees.
2. Pruning heuristics based on resource capacities to reduce the search space used by the proposed model without performance drawbacks
3. Techniques for configuring DSP application on the cloud-edge infrastructure with bandwidth guarantee.

1.5 Thesis Organization

In this current chapter (Chapter 1) we introduced key components to schedule DSP applications across cloud-edge infrastructures. In Chapter 2 we survey the state-of-the art, the open challenges and position our contribution on the topic. Based on the open challenges identified during the review of the state-of-the-art in Chapter 3 we present the problem statement addressed in this thesis and propose a system model with details on how optimization metrics are modeled as well as the operator parallelism [40, 41, 42]. Then, in Chapter 4 we present and evaluate an optimal placement solution based on the proposed scheduling model and compare it with standard solutions [41]. As an optimal model for the operator placement problem is likely to face scalability issues under large settings, Chapter 5 introduces a deployment solution also based on the scheduling model proposed in Section 3.3, but with a heuristic approach to overcome scalability issues that the optimal solution faces [42].

Chapter 2

Data Stream Processing Scheduling on Cloud-Edge Infrastructure

Contents

2.1 Introduction

Advances in Internet of Things (IoT) technologies are creating new challenges and opportunities for Data Stream Processing (DSP) applications where the data is generated by geographically distributed devices located at the edges of the network. In such scenarios, solutions focused on cloud deployment might not be ideal, despite the virtual unlimited resources that the cloud can provide. A recent trend consists in exploring resources at the edges of the network to offload parts of the processing from the cloud. Resources at the edge of the network include Micro Datacenters (MDs) and IoT devices [43].

In this chapter, we review scheduling solutions for DSP deployment and what elements they take into consideration, such as target infrastructure and its size, what challenges they address and how they are positioned in the literature. We present some of the frameworks for DSP, and review the state-of-the-art in DSP placement.

2.2 Background

DSP applications, depicted in Figure 2.1, are often long running. In the general workflow, a user of the application submits a deployment request containing the description of the application graph, the properties of the operators and their requirements. A DSP application graph is commonly structured as a Directed Acyclic Graph (DAG) whose vertexes are data sources, operators or data sinks, and edges represent how the data flows from sources, through operators, until it reaches the data sinks. A data source is responsible for receiving the data, sometimes connecting with an external system, and ingesting it into the application for processing. As the bottom part of Figure 2.1 depicts, this data may come from a sensor, an IoT device, another application, etc. The data sink is responsible for storing or providing the processed data to another application or user. DSP applications have at least one source and one sink, but commonly have more than that. Operators are responsible for processing the data stream by applying a pre-defined transformation (*i.e.*, it could be a filter, flat mapping, convolution, etc.) or any user-defined function.

An operator has multiple properties including *operator selectivity*, *data transformation factor*, and *operator state* [11, 25]. The operator selectivity refers to the number of messages the operator discards [44], and its selectivity could be classified as *selective* when the operator produces fewer messages than it receives, *one-to-one* when the operator produces the same number of messages that it receives, and *prolific* when the operator produces more messages than it consumes.

The data transformation factor reflects on how the operator changes the size of each message. It could be classified as *expansion* when the size of each message grows when compared to the size upon arrival, *stable* when the message size does not change, and *compression* when the operator reduces the size of the message.

The state of an operator refers to whether the operator maintains any information to process arriving messages. It can be further categorized under stateless, where the operator does not maintain any state, partitioned stateful where the operator maintains a key-based data structure to decide how to split processed data under downstream operators and stateful where there is no structured data but maintains some state for processing arriving messages.

Along with the application graph, the deployment request contains requirements, including Quality of Service (QoS) constraints and/or computing demands for each individual operator. QoS constraints are usually associated with throughput, application end-to-end latency or response time, and monetary costs. Since DSP are focused on near real-time processing of large amounts of data, throughput and application end-to-end latency are the main performance metrics. Requirements can comprise the number of replicas, CPU, memory, storage, etc. The correct assessment of requirements has a huge impact on the performance of the application, and usually they are done by the application owner/developer who has little or no expertise on how to perform these tasks [45].

The deployment request along with the requirements and the application graph are submitted to the *Application Scheduler* of a Infrastructure Service Provider (ISP) (Figure 2.1). Upon receiving the deployment request, the *Application Scheduler* tries to optimize the application graph towards the requirements, by creating multiple replicas for each operator and deciding how the load should be distributed between replicas. For instance, deciding how to share the load to create the required number of replicas, or how many replicas to create to guarantee a given response time or throughput. Parallelism can be explored in three ways [46]:

- *Task parallelism* where two different segments of the application graph process the same set of data at the same time. Since, only the application owner/developer has the knowledge to

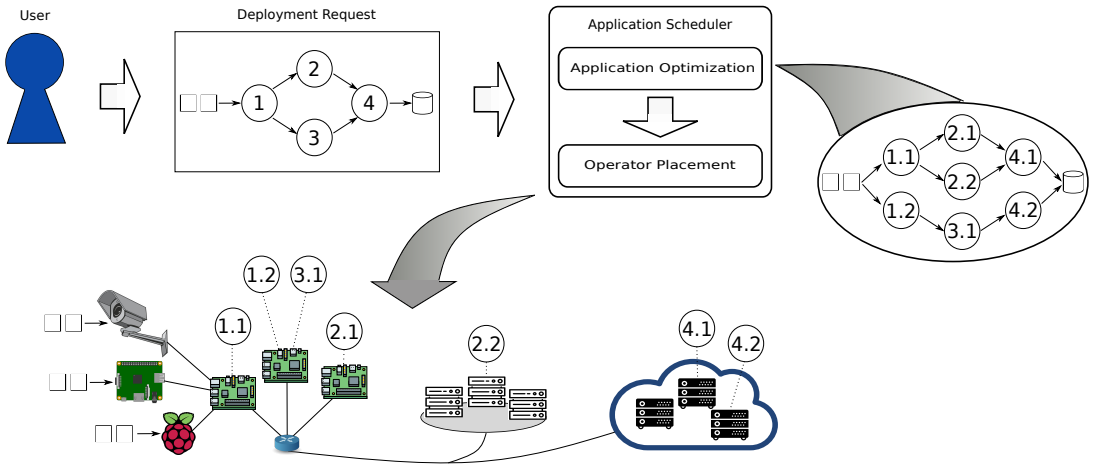


Figure 2.1: Overview of the deployment process of DSP applications on cloud-edge infrastructures.

determine if two segments of the application should process the same set of data, scheduling solutions do not explore *Task parallelism*.

- *Data parallelism* balance the load of a single operator among multiple replicas of it.
- *Pipeline parallelism* creates multiple instances of pipeline regions of the application and balances the load among these regions.

After optimizing the application, the solution needs to decide where and how to deploy operators or replicas within the physical infrastructure. This is known as the *operator placement problem*, which is NP-Hard [28], and consists in finding a subset of the physical infrastructure to host either the operators and/or their replicas. A solution for the *operator placement problem* aims to optimize one or multiple metrics and comprises an application model and an infrastructure model. The infrastructure model represents the target infrastructure, its complexity, and constraints that need to be met. For instance, in a model focused only on cloud deployment the network does not face as much congestion as in deployment on cloud-edge infrastructure where the communication between sites usually traverses the Internet without any bandwidth guarantees. The same goes for computing capabilities of resources; cloud servers are more powerful and pose much less constraints than edge resources.

2.3 Data Stream Processing Frameworks

Over the years, several DSP frameworks have been designed, such as Apache Storm that follows a master-worker architecture, depicted in Figure 2.2[1]. While the master is responsible for scheduling operators and handling failures, the work is deployed on numerous resources – physical or Virtual Machines (VMs). Each *Worker Node* can host multiple operators managed by a *Supervisor*. The number of operators in a worker is defined by the number of slots, which is an administrator-defined property. The actual number of replicas used by each operator of a DSP application is defined by the application owner.

[1]Figures 2.2, 2.3 and 2.4 are adapted from Dias de Assunção *et al.* [13]

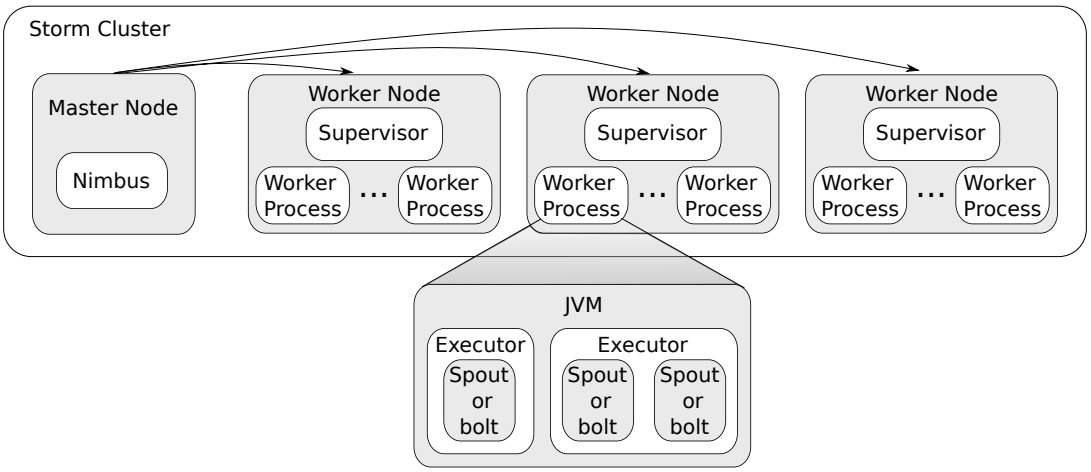


Figure 2.2: Apache Storm components

Twitter Heron works in a similar way to Apache Storm, with a master-worker architecture depicted in Figure 2.3. The master, also called *Topology Master*, is responsible for scheduling and handling workers. *Workers* are deployed in containers, where each container hosts: a *Stream Manager* responsible for forwarding messages; *Heron Instances* where the operators are deployed; and a *Metrics Manager* responsible for collecting execution and performance metrics. Twitter Heron differs from Apache Storm regarding the isolation for each operator, enabling a more fine-grained metric collection, used by the Topology Master, to apply a built-in back-pressure mechanism to adjust the number of messages processed by each operator instance. Despite having a back-pressure mechanism, Heron does not adjust the requirements provided by each operator instance, sometimes leading to under usage of resources and still requires application owners intervention to specify the number of instances for each operator.

Similar to Apache Storm, Apache Flink has the notion of slots, where the ISP defines the number of slots for each worker. The execution model for Apache Flink, Figure 2.4, is composed of a master called *JobManager* and workers named *Task Manager*. The *JobManager* is responsible to coordinate all the *Task Managers*, deploy operators, handle failures and many other management tasks, where the *Task Manager* host subtasks inside a Java Virtual Machine (JVM). The DSP application is broken into subtasks and partial streams, based on the parallel degrees of operators and streams defined by the applications owner/developer. Unlike other frameworks, Apache Flink has an efficient memory management mechanism that helps to adapt the requirements for each subtask in a fair and load-based fashion.

Moreover, in recent past years multiple data stream solutions have been conceived to run on constrained devices. Examples include: Apache Edgent [47], conceived by IBM and originally called Quarks [48]; lightweight versions of dataflow systems such as Apache MiNiFi [49]; data stream processing APIs such as Apache Kafka streams [50]; and software frameworks such as Node-RED [51] to interface with data collecting devices and design dataflows. Existing work has also attempted to create custom versions of Apache Storm able to run across multiple sites [52]. Most of these lightweight frameworks are designed to run on an individual resource, but multiple instances can be interconnected by using messaging systems such as message brokers or publish-subscribe software frameworks.

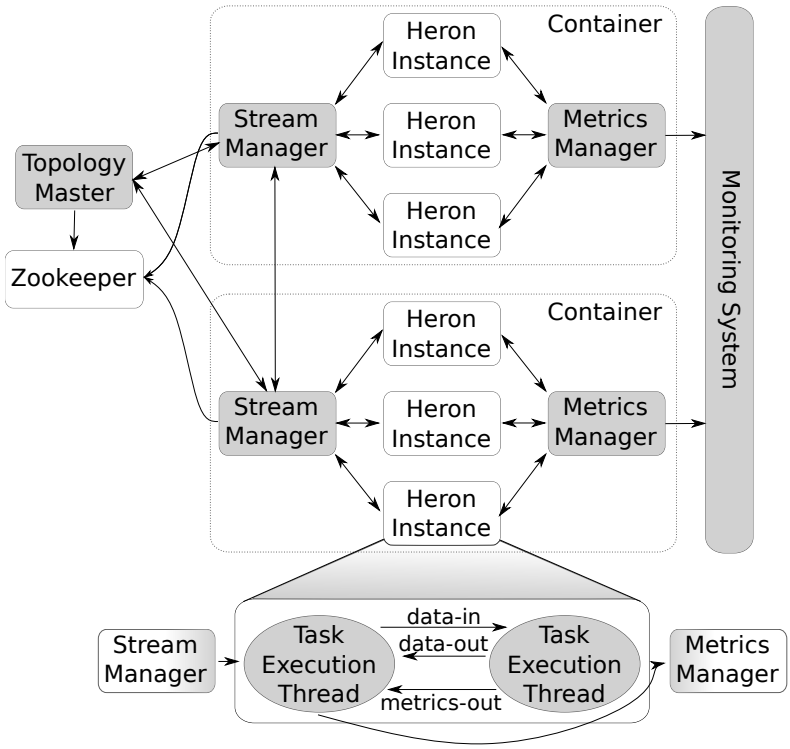


Figure 2.3: Twitter Heron components.

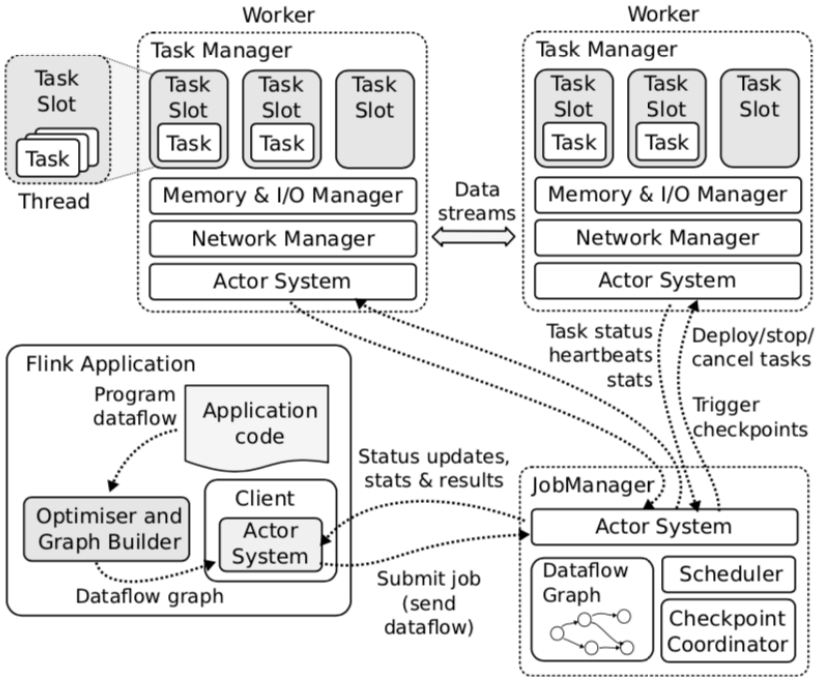


Figure 2.4: Apache Flink components.

2.4 Target Computing Infrastructure for DSP Placement

Frameworks and scheduling solutions are designed focused on a target infrastructure that has a major effect on performance metrics. On one hand, the use of edge resources can provide a performance boost for DSP applications in IoT scenarios. On the other hand, DSP applications for web analytics can be constrained to cloud resources with more simplistic application and infrastructure models since the network does not pose as many challenges as in the Internet. DSP applications for IoT scenarios can explore three organizational infrastructures: cloud, edge, and edge with IoT devices. This section discusses placement solutions targeted for DSP on cloud and edge computing environments. For the reader interested in a more general discussion on service placement and resource management aspects for cloud-edge infrastructure in general, we refer to references [53] and [54].

2.4.1 Data Stream Processing Solutions for Cloud Computing

There is a wide range of works in the literature that address deployment of DSP applications on cloud. Aljoby *et al.* [55] propose a solution to improve the performance of DSP applications through network reconfiguration imposing bandwidth guarantees via Software Defined Networking (SDN). The work evaluates the amount of data arriving at and leaving each operator, the direction towards which the data is flowing, and impose a fair share of bandwidth between flows crossing the same network path. The performance improvement obtained by Aljoby *et al.* is limited for two main reasons. First, it only addresses network reconfiguration, and secondly, the application model is very limited, which could create a large gap between real application and model. Zhang *et al.* [56] take a step further and account not only for the network reconfiguration but also for the operator placement, where operators are modeled considering only a resource requirement and the ratio between the volume of data arriving at and leaving the operator for each downstream. Given the application DAG, the information from the operator placement, and the information from the infrastructure, the solution proposed by Zhang *et al.* place operators aims to reduce the communication costs. They consider a more detailed operator model, but it optimizes deployment costs ignoring performance metrics such as bandwidth and application end-to-end latency, which are important for near real time processing. Moreover, the operator requirements and resource capacity are generic elements, and they do not address specific requirements in terms of CPU, memory, and storage requirements.

Liu and Buyya [45] address the placement problem with a more detailed resource model that considers CPU and memory, but with a simple operator model, composed only by the size of streams arriving at the operator. Their work considers deployment with pre-defined sizes of VMs or physical resources. To reduce deployment costs, their work aims to explore at maximum each resource, which reduces deployment costs and inter-node communication. The reduction of inter-node communication improves the application performance. Liu and Buyya also consider operator replication, but with equal load distribution between replicas. Although it is very interesting to use operator replication, their work considers a limited operator model, which might lead to inefficient deployment.

Gedik *et al.* [57] propose a scheduling solution for throughput optimization that considers partitioned stateful operators. The solution breaks the application graph into multiple regions that can be executed in parallel to achieve higher throughput. Parallel regions are evaluated, and they can trigger some replication of a region to maintain the throughput performance. Although the work proposed by Gedik *et al.* addresses several elements related to scheduling

problems, it still considers a very simple application and resource models that do not account for requirements and physical constraints. Hochreiner *et al.* [58] propose a solution that explores a generic placement and, through performance evaluation, adapts the application to reduce costs and the amount of used resources. Like other solutions, they consider a limited resource model that takes into account only CPU capacity, and an application model that does not consider the operators' requirements.

Tudoran *et al.* [59] propose a solution to improve the throughput of DSP applications, focused on scenarios where the data source of an application is in another cloud. Their solution creates micro batches of data that are sent from one cloud to another with less network interference. The size of the micro batch varies according with the network congestion, and multiple routes between clouds are explored to improve network transfer. By creating batches, the solution trades on maximizing throughput to small detriment of application end-to-end latency.

Eidenbenz and Locher [60] first provide mathematical proof that the operator placement problem is NP-Hard and then propose a placement solution assuming the DSP application as series of parallel dependent tasks that can be executed in parallel or a combination of such parallel series.

2.4.2 Data Stream Processing Solutions for Edge Computing

The advances in IoT technologies are creating scenarios where stream processing can leverage resources at the edge of the network. Cardellini *et al.* [11] explore a combination of IoT, MD, and cloud resources to deploy and reconfigure DSP applications to reduce deployment costs and response time on the graph critical path. The solution proposed by Cardellini *et al.* takes into consideration stateful operators and selects resources based on the requirements of each operator. The requirements for each operator are based on reference resources. Based on the application load the solution determines the number of replicas up to a maximum number. Peng *et al.* [22] focus only on edge resources, with a more detailed application and a resource with memory and CPU capacities, where the CPU capacity is expressed in Millions of instructions per second (MIPS). Operators have a probability of flow distribution for down streams, and the rate between the volume between arriving and after processing. The goal of Peng *et al.* work is to reduce deployment costs while guaranteeing a given response time. Beyond deployment, the work proposes a reconfiguration process based on bottleneck identification.

Aazam and Huh [23] proposed a solution for deployment and management on cloud-edge infrastructure. Their solution computes deployment costs and predicts the required resources on edge/MDs based on the risk of the application owner to release resources after leasing them. The rationale behind their approach is that resources at the MD are scarce. To maximize the profitability of the infrastructure owner, it either considers loyal users who would not release resources after leasing, or charge higher prices from risky users.

Taneja and Davy [61] propose a scheduling solution to explore both resources at the edge and at the cloud. It aims to reduce the application latency. It focuses on offloading operators from the cloud to the edge of the network. Then, for each operator, it iterates through all the resources at the edge of the network, trying to find a resource that fits the operator's requirements, in terms of CPU, memory and storage. If it does not find a resource at the edge, it searches for one among cloud resources.

Fu *et al.* [14] propose a data stream processing engine that considers resources at the edges of the network, such as IoT resources, sensors, gateways, amnong other resources. The proposed data processing engine, named EdgeWise, is operator-congestion aware and has a limited pool

of resources. Fu *et al.* argue that most processing engines create a mapping of one operator per thread, which in the cloud is not much of a problem due to high computational power of cloud resources, but with constrained resources such as the edge ones, this can create scenarios where threads with operators with a large processing queue are not scheduled frequently, affecting the performance of the whole application. In such scenario, EdgeWise has a thread pool and a hank of deployed operators ordered by the size of their processing queue, and at each time the operators with the highest queue size are scheduled to the thread pool, it increases the throughput and reduces end-to-end latency.

Zhang *et al.* [21] proposes a solution for the placement and reconfiguration of DSP applications exploring resources at the edge of the network. They argue that with the advances on IoT, devices deployed at the edges of the network it is more efficient to move computation than to move data. With that in mind, they proposed a solution that deploys operators onto edge resources exploring co-location of operators to reduce network interference. They also proposed a reconfiguration technique that creates a prediction model based on genetic algorithms that determine when the load of the application is going to increase. If the difference between the predicted load and the current load is higher than a given threshold it triggers a reconfiguration.

2.5 Mechanisms for Data Stream Processing Scheduling

Regardless the infrastructure that frameworks target, they need to decide where to deploy each operator and sometimes how to configure it in terms of allocated CPU, memory and, storage capacity. The deployment decision is commonly done using scheduling solutions. There are multiple ways to develop scheduling solutions with pros and cons on each of them.

2.5.1 Optimal Solutions

Cardellini *et al.* [11] propose a scheduling solution based on a linear formulation that aims to reduce deployment costs and the response time in a dataflow critical path. Their solution is also used to reconfigure DSP applications at runtime. The proposed model composes a multi-weighted objective function that, along with the deployment costs and response time in the critical path, accounts for the reconfiguration downtime. The model is also used to determine the number of replicas by using multi-sets, which are limited to a maximum number of replicas per operator. The solution proposed by Cardellini *et al.* evaluated multiple weights for each metric. As it is an optimal model based on linear formulation, it faces scalability issues due to the large distributed infrastructure of cloud-edge scenarios.

Amarasinghe *et al.* [25] claim that solutions that explore resource at the cloud-edge infrastructure should not just take into consideration performance metrics and computational requirements, but also the energy constraints that the use of an edge resource incurs. In this sense, Amarasinghe *et al.* proposed a deployment solution that tries to offload operators from the cloud, aiming to reduce the processing and transfer latency, while also ensuring that all the operators deployed inside the same edge resource would not demand more energy for processing than the power source available for the device can provide.

Hiessl *et al.* [62] argue that edge resources are dynamic and volatile in IoT scenarios. Such characteristics should be taken into consideration when deploying DSP applications. The work proposed by Hiessl *et al.* [62] considers multiple QoS metrics, among them the availability of edge resources, the end-to-end application latency, and deployment costs. The solution is based on Integer Linear Programming (ILP) with a multi-objective optimization function composed

of availability and end-to-end application latency. Also, the solution proposed by Hiessl *et al.* is designed to run over the infrastructure periodically and continually optimize DSP applications, then it also accounts for the migration costs of each operator.

According to Gu *et al.* [63], due to the virtually unlimited resources and elasticity, the cloud poses as a suitable solution for DSP solutions. However, ISP such as Amazon, Google, etc. usually run their solutions over multiple Datacenter (DC) infrastructures, which communicate over the Internet, introducing both network delay and monetary costs. The proposed solution aims to compose a virtual graph with sources and sinks and one replica per operator per DC infrastructure. After composing the virtual graph, the solution finds a placement solution to determine in which DC should be deployed replicas for each operator, the amount of data processed by each replica, focusing on reducing the network communication costs. The problem is modeled as a Mixed Integer Linear Programming (MILP), and to avoid scalability issues commonly faced by MILPs, and obtain solutions in polynomial time, integer variables are relaxed.

2.5.2 Heuristic-Based Solutions

The scalability issues imposed by optimal solutions and the large search space of cloud-edge infrastructure demand a sub-optimal, but more flexible approach. Zhang *et al.* [56] modeled the problem as a MILP, but explored a heuristic to relax and solve the model. The infrastructure model considered a multi-path network, and the goal was to reduce network costs. Despite the large search space, the solution proposed by Zhang *et al.* obtains deployment solutions in polynomial time. Peng *et al.* [22] also created a model based on linear formulation, with elements of queuing theory that compute each operator's load and service time. Beyond operator placement, the solution proposed by Peng *et al.* is applied to the reconfiguration though the bottleneck identification. The objective of the solution proposed by Peng *et al.* is to reduce deployment costs and guarantee a maximum response time through constraints, and to handle the scalability issue, it applies a binary generic algorithm to find a solution.

Despite not proposing a solution for operator placement Aljoby *et al.* [64] proposed a network reconfiguration solution for DSP applications. Aljoby *et al.* propose a heuristic that, based on the load of each stream of the application applies a fair share of the network bandwidth. Liu and Buyya [45] proposed a solution based on linear formulation. Based on the properties of the application and profiling information, it can determine the required number of replicas for load-balancing with the goal of reducing deployment costs and inter-node traffic. Due to scalability reasons Liu and Buyya develop a heuristic that is a variation of the *First Fit Decreasing* algorithm for solving the bin packing problem.

Truong *et al.* [65] propose a solution based on queueing theory to predict throughput and latency of DSP applications deployed on the cloud. Based on the expected load and on the prediction model, the solution of Truong *et al.* can determine the number of replicas for each operator and their respective placement. Eskandari *et al.* [66] aims to explore the data locality that edge resources can provide and the collocation of operators and their respective replicas, and therefore reduce end-to-end latency and increase throughput. The solution proposed by Eskandari *et al.* is composed of two steps: first it identifies highly communicating operators and groups them in such a way that they can fit available resources in the underlying infrastructure; second, for each group it finds the most communicating operators to run inside the same process, aiming to reduce the inter-process communication.

Elgamal *et al.* [31] recognize the benefits that resources with non-negligible computational power at the edges of the network can introduce to DSP applications. The high asymmetry

between cloud and edge resources, however, requires a model carefully elaborated to explore the trade-off between constrained computation with low latency and virtually unlimited computation with high latency. To this end, Elgamal *et al.* propose a model based on profiling estimates the response time of each operator, and with that it places operators with an algorithm based on dynamic programming to reduce end-to-end latency. Gedik *et al.* [44] propose a solution for the pipeline fission problem, that is automatically finding the best configuration of combined pipeline and data parallelism in order to optimize application throughput. The heuristic proposed on [44] aims to identify pipelined regions on the application graph that can be fused to be executed inside the same process without any queue between operators, and further fissioned, creating multiple replicas of such pipeline.

2.5.3 Machine-Learning Solutions

Cardellini *et al.* [52] propose a reconfiguration solution for DSP applications based on self-adaptation without any intervention from the application owner, and that explores the strength of both centralized and decentralized management solutions. The reconfiguration happens in two levels. First, at operator level, the performance of each operator is evaluated and reconfigured if necessary. At the second level, the whole application is evaluated, considering the communication and load distribution between operators. Both levels of reconfiguration use the Monitoring, Analysis, Planning and Execution (MAPE) architecture to decide when and how to reconfigure the operator or the whole application, and instead of using a strategy that triggers a reconfiguration based on threshold, the solution composes a reconfiguration cost metric based on reinforcement learning that adapts the evaluation based on the current state of the applications and avoids constant reconfiguration on the first level (operator level) without global knowledge of the application.

Da Silva Veith *et al.* [67] propose a reconfiguration solution based on reinforcement learning with a multi-objective optimization function. The work models the reconfiguration as a Markov Decision Process and explores two reinforcement learning solutions to find a solution. In the proposed solution the reconfiguration is triggered considering de MAPE architecture, and to avoid constant reconfiguration it takes into consideration the migration of operators and the time it requires for the applications to be paused before migrating.

Ni *et al.* [68] argues that a good resource allocation strategy should well balance the trade-offs between distributing computation evenly across devices and minimizing communication cost between devices. However, existing models either have strong assumptions of data arrival rate and node connectivity, or fail to fully capture the complex factors affecting the data processing throughput in practical stream processing systems. Then Ni *et al.* [68] propose a solution that explores Deep Reinforcement Learning to find an accurate model for the application and then propose a solution for the graph partitioning problem. The proposed solution aims to learn meta information from the application graphs that are generic and able to represent any application graph, creating a meta model that is trained just once, and with that information find a solution for the graph partitioning that is able to place operators into resources.

The work proposed by Russo *et al.* [69], claims that most solutions addressing the elasticity in stream processing applications assume a very simplistic view of the underlying infrastructure, which is often heterogeneous. Then, it addresses the problem of controlling elasticity of DSP applications deployed on heterogeneous infrastructure with the goal of minimizing the response time, resource usage and reconfiguration overhead. Their solution aims to create a infinity-horizon Markov Decision Process (MDP). MDP systems, however, have two major limitations,

Table 2.1: State-of-the-art solutions for the operator placement problem and reconfiguration of DSP applications.

Solution	Target Infrastructure	Metrics	Approach	Replication	Infrastructure Size
Aljoby *et al.* [55]	cloud	throughput	heuristic	no	10 resources
Zhang *et al.* [56]	cloud	monetary costs	heuristic	no	5 resources
Liu and Buyya [45]	cloud	monetary costs and throughput	heuristic	yes	3 resources
Gedik *et al.* [57]	cloud	throughput	heuristic	yes	4 resources
Cardellini *et al.* [11]	cloud-edge	monetary costs and end-to-end latency	optimal	yes	15 resources
Azam and Huh [23]	cloud-edge	monetary costs	optimal	no	resource estimation
Fu *et al.* [14]	cloud-edge	throughput and end-to-end latency	heuristic	no	1 resource
Zang *et al.* [21]	cloud-edge	network latency	heuristic	no	520 resources
Peng *et al.* [22]	edge only	monetary costs	heuristic	no	100 resources
Eskandari *et al.* [66]	edge only	throughput and end-to-end latency	optimal	no	8 resources
Elgamal *et al.* [31]	cloud-edge	end-to-end latency	optimal	no	2 resources
Gedik *et al.* [44]	cloud	throughput	optimal	yes	1 resource
Cardellini *et al.* [52]	cloud-edge	monetary costs	reinforcement learning	yes	8 resources
da Silva Veith *et al.* [67]	cloud-edge	monetary costs and end-to-end latency	reinforcement learning	no	42 resources
Ni *et al.* [68]	cloud	cost	deep reinforcement learning	no	5 resources
Russo *et al.* [69]	cloud	end-to-end latency	reinforcement learning	yes	10 resources

first it has a large search space that grows with the size of the infrastructure and its heterogeneity, and secondly an MDP requires a complete system knowledge, which sometimes is not possible at run-time.

2.6 Discussion and Positioning

Table 2.1 summarizes the solutions investigated in this chapter. Regarding the target infrastructure, the cloud still poses as a viable option because depending on the domain of the DSP application, the data source is at the cloud as well. The table also highlights works that explore edge computing either alone or in combination with the cloud. As mentioned beforehand, edge computing poses challenges that go beyond finding a placement for the application. As edge resources are computationally constrained, along with searching a placement for operators, a scheduling solution needs to find options to overcome such resource constraints by deciding how to run the operator code in the resource itselfs. An example is a work proposed for Fu *et al.* [14], that controls the scheduling decisions of the CPU to schedule threads or processes of operators that are with high processing queue. Or the work of Elgamal *et al.* [31] that decides which operators to offload to the edge with a queuing theory based model that evaluates the trade-off between computation and communication.

Another option to explore constrained edge resources is operator replication. Even though edge resources are computationally constrained they may be available in large numbers. A solution that creates multiple replicas for each operator and shares the load among such replicas might produce better results. This approach has been previously explored in the literature. Cardellini *et al.* [11] proposed a model to decide the number of replicas for each operator, but the number of replicas is limited to a given maximum. With a maximum number of replicas, there is a limitation to the load balancing among edge resources, which might overload replicas. Liu and Buyya [70] propose a solution to create replicas, but assume an even load distribution for each replica on a heterogeneous infrastructure, which might overload certain replicas.

DSP applications are long-running and designed to process large amounts of data in a short time. In summary, it explores the trade-off between using powerful computational resources from the cloud and low latency resources from the edge, with a performance goal that maximizes the processing time or maximizes the amount of processed data. This is reflected in the literature, where throughput and end-to-end latency are very common metrics as shown in Table 2.1, either for cloud or cloud-edge solutions. Since most of theses solutions explore some type of public infrastructure, metrics such as monetary cost are very common because companies and organizations usually have limited budgets.

In order to find placement solutions, existing work usually explores heuristic solutions as demonstrated in the table. We believe that the reason is the scalability issues that optimal solutions often impose, and the fact that a cloud-edge infrastructure can be very large with geographically distributed resources. Despite this fact, some efforts towards optimal solutions in the cloud-edge have been made, such as Eskandari *et al.* [66] and Cardellini *et al.* [11]. However, most approaches usually evaluate the proposed solutions in small-sized environments that do not reflect the reality of cloud-edge infrastructure. The same holds true for heuristic approaches, except for Zang *et al.* [21] that account for an infrastructure with 520 resources, but 480 of these are sensors that are not considered as candidates to host operators.

This thesis proposes an optimal model to schedule DSP applications exploring cloud-edge resources, that accounts for the heterogeneity of the target infrastructure, the network limitations connecting edge and cloud, and the characteristics of the application and its operators, such as

stream patterns, operator selectivity, data transformation factor, etc. By coping with network and resource limitations from edge resources, we explore bandwidth guarantees and operator replication with the load processed by each replica decided by the model, and without limiting the number of replicas.

2.7 Conclusion

The operator placement problem has several inherent challenges. Although some of them have already been addressed by the community, there is still space for improvement or other approaches to evaluate. For instance, some solutions address the operator replication, but often the number of replicas needs to be defined by the application owner/developer. Sometimes the model itself determines the number of replicas, but it is limited to a maximum number or the load among replicas is uniformly distributed. In such scenarios, despite operator replication being addressed there is yet much to be explored. We have shown the pros and cons of focusing either on cloud infrastructure or on cloud-edge infrastructures, and the different ways to solve the placement problem.

Among the open challenges, we investigate a solution for the operator placement problem for DSP applications focused on IoT scenarios, exploring the benefits of the edge and the cloud, with operator replication on constrained devices such as IoT ones, network configuration with bandwidth guarantees and requirements estimation to reduce response time and deployment costs.

Chapter 3

Mixed-Integer Programming Model for Data Stream Processing

Contents

3.1 Introduction

As presented in Chapter 2, there are several solutions for addressing the placement and parallelism of Data Stream Processing (DSP) applications. Despite the fact that a large part of these solutions focus on cloud computing, there is a growing demand for exploiting edge computing resources or the combination of cloud and edge computing resources to improve the Quality of Service (QoS) and reduce incurred costs. The work that focuses on the cloud-edge infrastructure provides new opportunities for deploying DSP applications on heterogeneous environments, but still many issues need to be addressed, such as devising more detailed models of the infrastructure considering the CPU, memory, and storage capacities of resources, and a more precise application model that takes into consideration operator characteristics such as selectivity, data transformation pattern, among other factors. Another aspect is that existing work either overlooks or is limited in the manner it addresses operator replication; an essential feature when dealing with constrained devices such as those provided by edge computing. The few solutions

that address operator replication, do so assuming an even load distribution between replicas, or have a cap to the number of replicas created, hence limiting the solution's performance.

This chapter introduces models for DSP applications, cloud-edge infrastructure, and performance metrics. The application model includes elements such as selectivity, data transformation pattern, and probability of load distribution. The infrastructure model takes into consideration the heterogeneity of edge resources, with different computational power modeled as the clock speed of each resource. Also, the application model explores operator parallelism by splitting an operator into smaller replicas to fit edge resources, and provides the requirements according the load that each replica should process, which might not be even for all replicas. The load of each replica is determined considering the network latency and congestion and the capacity of the resources into which they are deployed. The model aims to find a solution to reduce the application end-to-end latency, by reducing the network interference on data transfer with bandwidth guarantees, and deployment costs. We employ a cost model based on elements from AWS Fargate Pricing [71] and AWS Direct Connection [72]. Application end-to-end latency is a very important metrics in many of the current and emerging DSP scenarios.

3.2 Deployment Architecture

An architecture to assist deployment solutions and ease the process of scheduling DSP applications onto cloud-edge infrastructure is depicted in Figure 3.1. The architecture comprises modules to manage and extract information from the physical resources and a deployment engine for DSP applications. We envision that this architecture can be realized using existing software frameworks for DSP applications and message-brokering systems as later demonstrated in Section 3.4.2, which evaluates a throughput model. Moreover, the operator placement and replication techniques can be incorporated into existing DSP frameworks, such as a custom scheduler for Apache Storm.

The computational resources follow the three-layered structure of cloud-edge described beforehand: Internet of Things (IoT) resources, Micro Datacenters (MDs), and cloud computing. On each computing resource there is an instance of a *Node Manager*, in turn composed of two modules, namely the *Operator Manager* and *Performance Monitor*. While the *Operator Manager* is responsible for managing operators of DSP applications deployed on the resource – including deploying, releasing, and scaling operators – the *Performance Monitor* collects performance metrics both from the resource and each operator instance, and reports the metrics to the *Resource Manager* in the *Data Stream Processing Engine*, a centralized module responsible for managing the computing resources. The *Resource Manager* combines the performance metrics collected from the *Performance Monitor* module from each resource, with network information obtained via techniques such as Vivaldi [73] or Software Defined Networking (SDN) [74] used to discover the network topology and build an infrastructure graph with the capacity and availability of each computing resource and the network, as well as, impose network guarantees.

The *Dataflow API* is utilized by users to interface with the *Data Stream Processing Engine* and submit a deployment request for a DSP application. A deployment request contains the description of the DSP application graph, the requirements and properties of each operator, and the QoS requirements, which can be throughput maximization, application end-to-end latency minimization, cost minimization, or a combination thereof. The application and its requirements are then passed to the *Application Scheduler*, which is continuously provided with information gathered by the *Resource Manager* on available resources, their residual capacity, their network interconnections, and the network capacity. Along with information from the DSP application

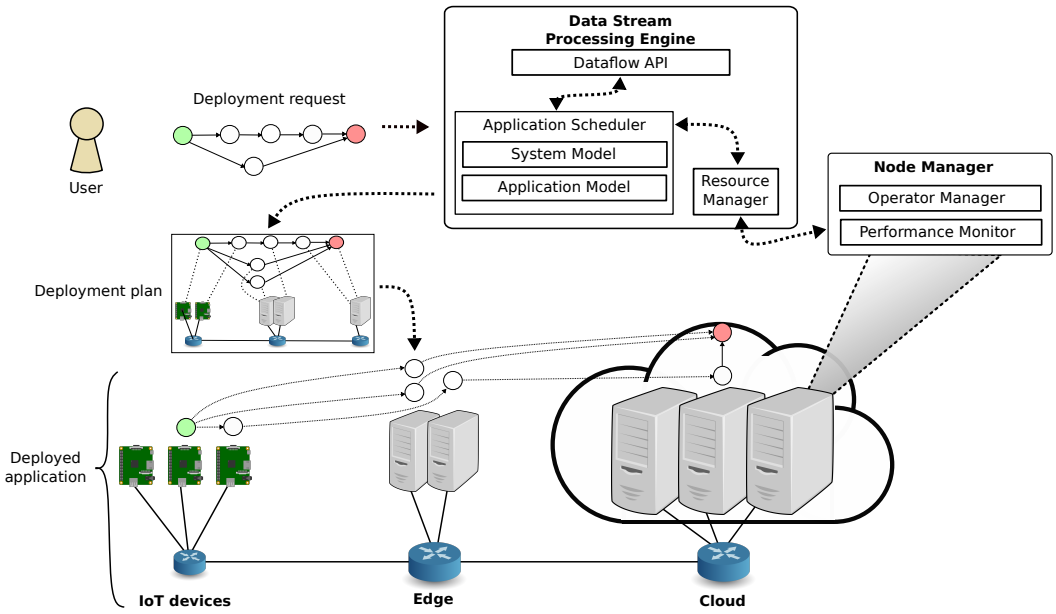


Figure 3.1: Overview of the architecture for deployment of DSP applications on cloud-edge infrastructures.

and the infrastructure the *Application Scheduler* devises a *schedule/deployment plan* that specifies how many instances of each operator must be created and on which resources these instances ought to be placed.

In order to compute a deployment plan, a scheduling algorithm often needs to work with models that describe how a system or application might behave in practice once it is deployed on the actual infrastructure. Such models are important to ensure that one or multiple QoS requirements for a DSP application can be met when the application is executed on the underlying system. Hence, the quality of the scheduling decisions computed by the *Application Scheduler* strongly depend on the accuracy of the system and application models it uses. As depicted in Figure 3.1, *Application Scheduler* uses the infrastructure and application models to identify how many instances of an operator must be created and where they must be placed in order to optimize the required performance metrics.

Once the *Application Scheduler* finds a deployment plan with the description of resources where each operator or replica should be placed, the requirements of each operator and the required network configuration, this plan is passed to the *Resource Manager* that coordinates with *Node Managers* and employs SDN techniques to enforce the network configuration. The DSP application can then start its execution.

3.3 System Model

This thesis considers a three-layered cloud-edge infrastructure, as depicted in Figure 3.2, where each layer contains multiple sites. The *IoT* layer contains numerous geo-distributed constrained computing resources often acting as data sources or sinks, but with non-negligible computational capacity to support some DSP operators. *MDs* provide geographically distributed resources (*e.g.*, routers, gateways, and micro datacenters), but with less computational constraints than those in the IoT layer. The *cloud* comprises high-end servers with fewer resource constraints [75]. The

notation used through this thesis is summarized in Table 3.1.

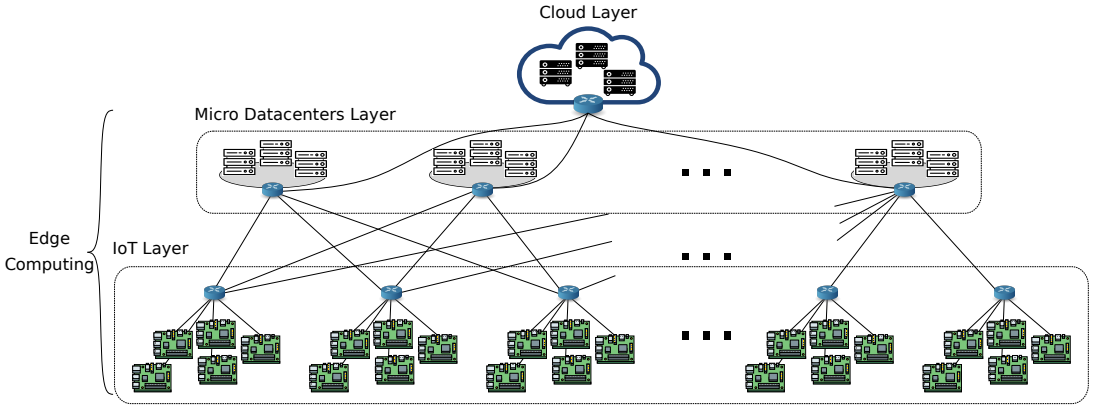


Figure 3.2: Three-layered cloud-edge computing infrastructure.

3.3.1 Infrastructure Model

The three-layered cloud-edge infrastructure is represented as a graph $\mathcal{G}^I = \langle \mathcal{R}, \mathcal{P} \rangle$, where $\mathcal{R}$ is the set of computing resources of all layers ($\mathcal{R}^{IoT} \cup \mathcal{R}^{MD} \cup \mathcal{R}^{cloud}$), and $\mathcal{P}$ is the set of network interconnections between computing resources. Each $k \in \mathcal{R}$ has CPU (CPU_k) and memory (Mem_k) capacities, given respectively in $100 \times num\_of\_cores$, and bytes. The processing speed (V_k) of a resource k is its CPU clock in GHz. Similar to existing work [29], the network has a single interconnection between a pair of computing resources k and l, and the bandwidth of this interconnection is given by $Bw_{k,l}$ and its latency is $Lat_{k,l}$.

3.3.2 Application Model

The application graph specified by a user is a directed graph $\mathcal{G}^A = \langle \mathcal{O}, \mathcal{E} \rangle$, where $\mathcal{O}$ represents data source(s) $Source^{\mathcal{O}}$, data sink(s) $Sink^{\mathcal{O}}$ and transformation operators $Trans^{\mathcal{O}}$, and $\mathcal{E}$ represents the streams between operators, which are unbounded sequences of data (*e.g.*, messages, packets, tuples, file chunks) [11]. The application graph contains at least one data source and one data sink. Each operator $j \in \mathcal{O}$ is the tuple $\langle S^j, C^j, \mathcal{U}^j, AR^j \rangle$, where S^j is the selectivity (message discarding percentage), C^j is the data transformation factor (how much it increases/decreases the size of arriving messages), $\mathcal{U}^j$ is the set of upstream operators directly connected to j, and AR^j is the input rate in bytes/s that arrives at the operator. When operator j is a data source (*i.e.*, $j \in Source^{\mathcal{O}}$) its input rate is the amount of data ingested into the application since $\mathcal{U}^j = \emptyset$. Otherwise, AR^j is recursively computed as:

$$AR^j = \sum_{i \in \mathcal{U}^j} \rho^{i \to j} \times DR^i \tag{3.1}$$

where $\rho^{i \to j}$ is the probability that operator i will send an output message to operator j, which captures how operator i distributes its output stream among its downstream operators. DR^i is the departure rate of operator i, given by:

$$DR^i = AR^i \times (1 - S^i) \times C^i \tag{3.2}$$

Table 3.1: Summary of notation used in this thesis.

Symbol	Description
$\mathcal{G}^I$	Infrastructure graph
$\mathcal{R}$	Set of computing resources ($\mathcal{R}^{IoT} \cup \mathcal{R}^{MD} \cup \mathcal{R}^{cloud}$)
$\mathcal{P}$	Set of paths between computing resources
p	A path p that belongs to $\mathcal{P}$
p_s	Source of path p
p_d	Destination of path p
CPU_k	CPU capacity of resource k
Mem_k	Memory capacity of resource k
V_k	Clock speed of resource k
$Bw_{k,l}$	Bandwidth of the path between resources k and l
$Lat_{k,l}$	Network latency of the path between resources k and l
$\mathcal{G}^A$	The DSP application graph
$\mathcal{O}$	Set of operators
$\mathcal{E}$	Set of streams between operators
$Source^{\mathcal{O}}$	Subset of sources from $\mathcal{O}$
$Sink^{\mathcal{O}}$	Subset of sinks from $\mathcal{O}$
$Trans^{\mathcal{O}}$	Subset of transformation operators from $\mathcal{O}$
S^j	Selectivity of operator j
C^j	Data transformation factor of operator j
$\mathcal{U}^j$	Subset of operators that send data to j
AR^j	Byte arrival rate of operator j
GR^j	Data generation rate of source operator j
$\rho^{i \to j}$	Probability that operator i sends data to operator j
DR^j	Byte departure rate of operator j
Req^j_{cpu}	CPU requirements to process the data arriving at operator j
Req^j_{mem}	Memory requirements to process the data arriving at operator j
Req^j_{bytes}	Requirements in bytes/s to execute operator j
Ref^j_{data}	Ref. data volume (bytes/s) processed to obtain Ref^j_{cpu} and Ref^j_{mem}
Ref^j_{cpu}	Reference values of CPU usage for operator j to process Ref^j_{data}
Ref^j_{mem}	Reference values of memory usage for operator j process Ref^j_{data}
Ref^j_V	Processing speed of reference resource where j was evaluated
Ω_j	Speedup/slowdown w.r.t. the ref. clock V and the clock of resource k (V_k)
$x(j,l)$	Amount of data that operator j processes on resource l
$f(i,k \to j,l)$	Amount of data flowing from operator i to j deployed on resources k and l
$C_{cpu}(l)$ and $C_{mem}(l)$	Cost per CPU unit and cost of storing one byte in memory at resource l
$C_{bw}(k,l)$	Cost of transferring a byte over the network from resource k to l
CC and NC	Computational and network costs
ATT	Aggregate data Transfer Time
β	Variation in the processing requirements of an application
$\mathcal{L}$	Layers of the cloud-edge infrastructure
$Source^{\mathcal{L}}$	Set of operators that receive data from operators in other infrastructure layers
$Sink^{\mathcal{L}}$	Set of operators that send data from operators in other infrastructure layers
ℓ^j	Infrastructure layer in which operator j is deployed
$PT(j)$	Processing time for sink operator j
$PP(j)$	Higher processing time path for sink operator j
$AR(j,l)$	Arrival rate of operator j deployed on resource l
$DR(j,l)$	Departure rate of operator j deployed on resource l

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

The quality of a placement is guaranteed by meeting the application requirements. The CPU and memory requirements of each operator j for processing its incoming byte stream are expressed as Req^j_{cpu} and Req^j_{mem} and they are obtained by profiling the operator on a reference resource [76]. Ref^j_{cpu}, Ref^j_{mem} and Ref^j_{data} refer to the reference CPU, memory and processed data of operator j, respectively. Since CPU and memory cannot be freely fractioned, the reference values are rounded up and combined with AR^j of j in order to compute Req^j_{cpu} (Equation 3.3) and Req^j_{mem} (Equation 3.4) that handle the arriving data stream.

$$Req^j_{cpu} = \left\lceil \frac{Ref^j_{cpu} \times AR^j}{Ref^j_{data}} \right\rceil \tag{3.3}$$

$$Req^j_{mem} = \left\lceil \frac{Ref^j_{mem} \times AR^j}{Ref^j_{data}} \right\rceil \tag{3.4}$$

3.3.3 Metrics Model

The rate of data ingested by sources is constant and stable, hence making it possible to compute the CPU and memory requirements recursively for the entire application to handle the expected load. Placing an application onto computing resources incurs a cost. By taking a closer look at services provided by a major Infrastructure Service Provider (ISP) such as Amazon, multiple services would be required to deploy a DSP application on IoT scenarios with operator isolation and bandwidth guarantees. To deploy operators with some level of performance isolation and with performance guarantees, the first service required would be Amazon AWS Fargate [71], that is a serverless solution to deploy containers with memory and CPU guarantees. As we consider IoT scenarios, the second service would be Amazon AWS IoT Core [77], that is a service to manage and collect data from IoT resources to the AWS infrastructure. Last, a deployment needs network guarantees, which is where Amazon AWS Direct Connect [72] and Amazon AWS Private Links [78] come into play. Amazon AWS Private Links [78] is a service where Amazon provides a private link with bandwidth guarantees between their infrastructure and the client's infrastructure. In the context of this thesis, these services could be used to realise the network interconnection between IoT resources and MD/cloud. Also, despite both being considered public infrastructure here, MD and cloud resources are on distinct geographical locations. Amazon AWS Direct Connect [72] is a service that provides bandwidth guarantees between other services deployed inside Amazon's infrastructure. The cost of using one unit of CPU and storing one byte in memory at resource l is given by $C_{cpu}(l)$ and $C_{mem}(l)$, respectively, while the cost of transferring a byte over the network from resource k to l is denoted by $C_{bw}(k,l)$.

As cloud-edge infrastructure comprises heterogeneous resources, the model applies a coefficient $\Omega_l = Ref^j_V / V_l$ to adapt the operator requirements to resource l. Ref^j_V is the reference processing speed of the resource for operator j, and V_l is the clock speed of resource l. The computational cost is given by:

$$CC = \sum_{l \in \mathcal{R}} \sum_{j \in \mathcal{O}} \frac{C_{cpu}(l) \times \frac{\frac{Req^j_{cpu}}{\Omega_l} \times \beta \times x(j,l)}{AR^j}}{\max C_{cpu}(l)} + \frac{C_{mem}(l) \times \frac{Req^j_{mem} \times x(j,l)}{AR^j}}{\max C_{mem}(l)} \tag{3.5}$$

where $\max C_{cpu}(l)$ and $\max C_{mem}(l)$ are the cost of using all the CPU and memory capacity of resource l. The CPU and memory costs are normalized using their maximum amounts resulting in values between 0 and 1. β refers to a safety margin to each replica requirements aiming to a steady safe system. This margin relies on queueing theory premises to avoid an operator reaching the CPU limits of a given computing resource, which requires a higher queuing time.

The network cost NC is computed as:

$$NC = \sum_{p \in \mathcal{P}} \sum_{a,b \in p} \sum_{j \in \mathcal{O}} \sum_{i \in \mathcal{U}^j} \frac{C_{bw}(a,b) \times f(i, p_s \rightarrow j, p_d)}{\max C_{bw}(a,b)} \tag{3.6}$$

where a, b is a link that represents one hop of path p, and a, b can belong to multiple paths. The resources at the extremities of path p hosting replicas i and j are given by p_s and p_d, respectively. NC is normalized by $\max C_{bw}(a,b)$, the cost of using all the bandwidth available between resources a and b.

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

$$ATT = \sum_{p \in \mathcal{P}} \sum_{k,l \in p} \sum_{j \in \mathcal{O}} \sum_{i \in \mathcal{U}^j} \frac{f(i, p_s \rightarrow j, p_d) \times (Lat_{k,l} + \frac{1}{Bw_{k,l}})}{Lat_{k,l} + 1} \tag{3.7}$$

The multi-objective function aims at minimizing the data transfer time and the application deployment costs:

$$\min : ATT + CC + NC \tag{3.8}$$

3.3.4 Infrastructure and Application Constraints

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

The objective function is subject to:

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

$$CPU_l \geq \sum_{j \in \mathcal{O}} \frac{\frac{Req^j_{cpu}}{\Omega_l} \times \beta \times x(j,l)}{AR^j} \tag{3.9}$$

$$Mem_l \geq \sum_{j \in \mathcal{O}} \frac{Req^j_{mem} \times x(j,l)}{AR^j} \tag{3.10}$$

The following guarantees that the amount of data traversing every link a, b does not exceed its bandwidth capacity:

$$\sum_{j \in \mathcal{O}} \sum_{i \in \mathcal{U}^j} f(i, p_s \to j, p_d) \leq Bw_{a,b} \qquad \forall a, b \in p; \forall p \in \mathcal{P} \tag{3.11}$$

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

$$AR^j = \sum_{l \in \mathcal{R}} x(j, l) \qquad \forall j \in \mathcal{O} \tag{3.12}$$

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

$$DR^i \times \rho^{i \to j} = \sum_{k \in \mathcal{R}} \sum_{l \in \mathcal{R}} f(i, k \to j, l) \qquad \forall j \in \mathcal{O}; \forall i \in \mathcal{U}^j \tag{3.13}$$

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

$$x(i, k) \times (1 - S^i) \times C^i \times \rho^{i \to j} = \sum_{l \in \mathcal{R}} f(i, k \to j, l) \qquad \forall k \in \mathcal{R}; \forall j \in \mathcal{O}; \forall i \in \mathcal{U}^j \tag{3.14}$$

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

$$\sum_{i \in \mathcal{U}^j} \sum_{k \in \mathcal{R}} f(i, k \to j, l) = x(j, l) \qquad \forall j \in \mathcal{O}; \forall l \in \mathcal{R} \tag{3.15}$$

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

$$x(j, l) = AR^j \qquad \forall j \in Source^{\mathcal{O}} \cup Sink^{\mathcal{O}}; \forall l \in \mathcal{R} \tag{3.16}$$

$$x(j, l) \geq 0 \qquad \forall j \in Trans^{\mathcal{O}}; \forall l \in \mathcal{R} \tag{3.17}$$

$$f(i, k \to j, l) \geq 0 \qquad \forall k, l \in \mathcal{R}; j \in \mathcal{O}; i \in \mathcal{U}^j \tag{3.18}$$

3.4 Throughput Estimation Model

Although the main contribution of this thesis is a model and algorithms for computing solutions to the operator placement and parallelism problems on cloud-edge infrastructure, this chapter also describes a model that can be used to estimate the throughput of DSP applications onto cloud-edge infrastructure. The goal of this model is to estimate the throughput that a deployment plan can achieve under certain environmental and application conditions.

3.4.1 Scenario and Model Description

The solution to the operator placement and parallelism problem consists of finding a *deployment plan* that establishes how many replicas of each operator must be created and onto which resources these replicas should be deployed. This section introduces a model for estimating the throughput that a given deployment plan for DSP applications on cloud-edge computing can achieve. As the throughput model and the scheduling model presented in Section 3.3 are interrelated, some equations and functions look similar. However, unlike the scheduling model that initially deals with the logical application graph specified by the user and aims to devise the physical graph for deployment, the throughput model deals already with the physical graph whose placement has already been decided by the scheduling. Hence, for instance, while AR_j represents the byte arrival rate of operator j in the logical graph, the throughput model uses $AR(j, l)$ to represent the byte arrival rate of operator j deployed on resource l. The notation used to describe the model is summarized in Table 3.1.

The estimation model follows both the infrastructure and the application models presented in Section 3.3. However we add another representation of an operator's requirements, where instead of requesting a given amount of CPU and memory, we consider that theses values are derived from a processing rate requirement Req^{j}_{bytes} given in bytes/s that operator j should process.

The proposed model considers that a deployment plan follows the three-layered structure of the cloud-edge computing infrastructure and hence splits the set of operators $\mathcal{O}$ into three deployment sequences, one for each layer $\mathcal{L}$ of the cloud-edge infrastructure (*i.e.*, IoT, MD, *cloud*) as depicted in Figure 3.3. This division into sequences is used to compute the throughput in each layer and to account for the network interference between layers. As it is essential to understand how operators deployed in one layer interact with operators in other layers, we use $Source^{\mathcal{L}}$ and $Sink^{\mathcal{L}}$ to represent, respectively, the set of operators of layer $\mathcal{L}$ that receive data from operators in others layers — or from *data sources* in the case of $\mathcal{L} = IoT$ — and the operators of $\mathcal{L}$ that stream data to downstream operators in other layers; or to the application *data sinks* for operators hosted on the *cloud*. The layer of an operator j is given by ℓ^j.

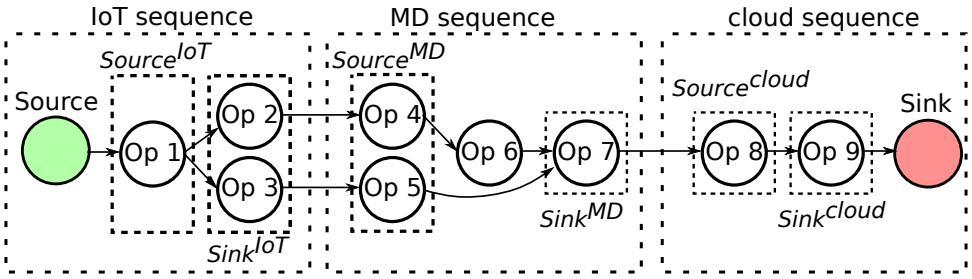


Figure 3.3: Deployment sequences for each layer of a cloud-edge infrastructure.

We assume that data sources are always deployed on resources at the IoT layer. Hence, for operators in $Source^{IoT}$ the arrival rate $AR(j,l)$ of operator j deployed on resource $l \in \mathcal{R}$ is equal to the data generation rate GR^j in bytes/s, which, as explained beforehand, is computed based on the number and size of messages ingested into the application. For the remaining operators, the $AR(j,l)$ of j on resource l is computed using Equation 3.19 that considers the parallelism level of operator j (*i.e.*, how many operator instances/tasks are executed) and the departure rate of all upstream operators $\mathcal{U}^j$ that send data to j:

$$AR(j,l) = \frac{\sum_{i \in \mathcal{U}^j} \sum_{k \in M(i)} \mathcal{S}(i,k \to j,l)}{|M(j)|} \tag{3.19}$$

where $M(j)$ returns a mapping $\langle$*operator, resource*$\rangle$ for all the deployed instances of operator j so that the number of replicas of j is given by $|M(j)|$. $\mathcal{S}(i,k \to j,l)$ is the byte rate that each upstream operator i deployed on resource k sends to operator j deployed on l. $\mathcal{S}(i,k \to j,l)$ is computed by Equation 3.20 as follows. If j and a previous operator i are on the same resource, only the departure rate $DR(i,k)$ of operator i on resource k is considered to compute the byte rate. Otherwise, it takes into account the departure rate of all operators in $M(i)$. Moreover, since i could send its output stream to various operators, the probability $\rho^{i \to j}$ that i forwards its outgoing stream to j is used again here.

$$\mathcal{S}(i,k \to j,l) = \begin{cases} \rho^{i \to j} \times DR(i,k) & \text{if } k = l \\ \min\left[\frac{\rho^{i \to j} \times DR(i,k)}{Max(PT(i),1)}, Bw_{k,l}\right] & \text{otherwise} \end{cases} \tag{3.20}$$

The departure rate $DR(i,k)$ of operator i onto resource k is given by Equation 3.21, which applies the selectivity S^j and data transformation pattern C^j on the arrival rate, changing the number and size of messages in the income stream to produce the output stream. The departure rate is also given in bytes/s.

$$DR(j,l) = AR(j,l) \times (1 - S^j) \times C^j \tag{3.21}$$

The application graph contains several paths between data sources and data sinks, and they can be decomposed into pipelines executed in parallel. To compute the throughput of each layer $\mathcal{L}$ considers the paths between $Source^{\mathcal{L}}$ and $Sink^{\mathcal{L}}$, where the throughput of each operator $j \in Sink^{\mathcal{L}}$ is computed considering the path with greatest processing time from $Source^{\mathcal{L}}$ to the operator itself. For instance, in the application in Figure 3.3 the throughput of MD layer is based on the throughput of Op 7, and the paths to this operator are Op 4 $\to$ Op 6 $\to$ Op 7 and Op 5 $\to$ Op 7. The path with the greatest processing time is picked to compute the throughput. This process of identifying and selecting the path with the greatest processing time, from the $Source^{\mathcal{L}}$ to $j \in Sink^{\mathcal{L}}$, is represented by $PP(j)$.

After selecting the path with greatest processing time, the model computes the time difference between the departure timestamp of two consecutive messages, from the last operator on the path. Let us assume that for the throughput of the MD layer of the application in Figure 3.3 the path with greates processing time is Op 4 $\to$ Op 6 $\to$ Op 7; the pipeline execution of this path is depicted in Figure 3.4. Message $M1$ arrives at Op 4 and is processed during 1 time unit, after which it is handed to Op 6. While Op 6 starts to process $M1$, Op 4 starts to process $M2$. After 1 time unit Op 4 finishes $M2$ and stores it in a queue while Op 6 finishes processing $M1$.

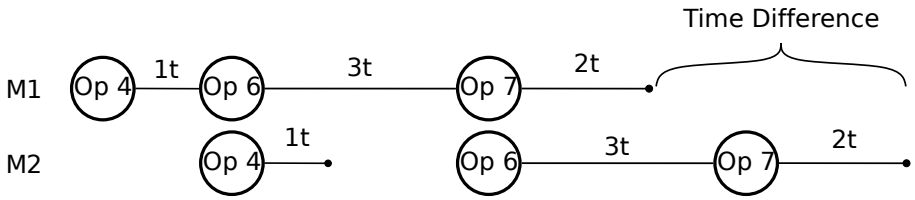


Figure 3.4: Difference between the finish processing times of two consecutive messages.

Both messages leave the path at $Op\ 7$, so that we compute the time difference with the departure timestamp from this operator. This time difference is given by $PT(j)$ (Equation 3.22), which is used to compute the throughput of operator j. The model computes this time difference for the arrival rate of the first operator of the path, whereby if this time difference is lower than one second, operator j can process more than the arrival rate. However, since the throughput is the departure rate produced in one second, we consider one second as the minimum time difference between two consecutive messages.

$$PT(j) = \frac{AR(first(PP(j)), M(first(PP(j))))}{Req_{bytes}^{first(PP(j))}} + Q(j) - \sum_{i \in PP(j)} \frac{AR(i, M(i))}{Req_{bytes}^{i}} \tag{3.22}$$

where $Q(j)$ is given by:

$$Q(j) = \sum_{i \in PP(j)} \begin{cases} \frac{AR(i,M(i))}{Req_{bytes}^{i}} & i = last(PP(j)) \\ \max\left[\frac{AR(i,M(i))}{Req_{bytes}^{i}}, \frac{AR(i+1,M(i+1))}{Req_{bytes}^{i+1}}\right] & \text{otherwise} \end{cases} \tag{3.23}$$

The proposed model computes the throughput of each layer $\mathcal{L}$ (Equation 3.24) based on the departure rate of all operators $j \in Sink(\mathcal{L})$ and the time difference $PT(j)$. Since the throughput is based on the departure rate, it is given in bytes/s. The throughput of the DSP application is given by $Rate(cloud)$:

$$Rate(\mathcal{L}) = \sum_{j \in Sink(\mathcal{L})} \frac{DR(j, M(j))}{\max(PT(j), 1)} \tag{3.24}$$

3.4.2 Experimental Setup

To evaluate the proposed throughput model we use a real testbed as our cloud-edge environment on which we deploy a DSP application. We compare the throughput obtained by the model against the throughput achieved by deploying the application on the testbed.

Infrastructure

The cloud-edge infrastructure testbed depicted in Figure 3.5 is organized as follows. The IoT layer comprises 2 Raspberry PI's 3 (*i.e.*, ARMv7 at 1.2 GHz and 1 GB of RAM), both connected to a gateway via a 100Mb/s network and latency of 0.4ms [29]. The gateway is a server

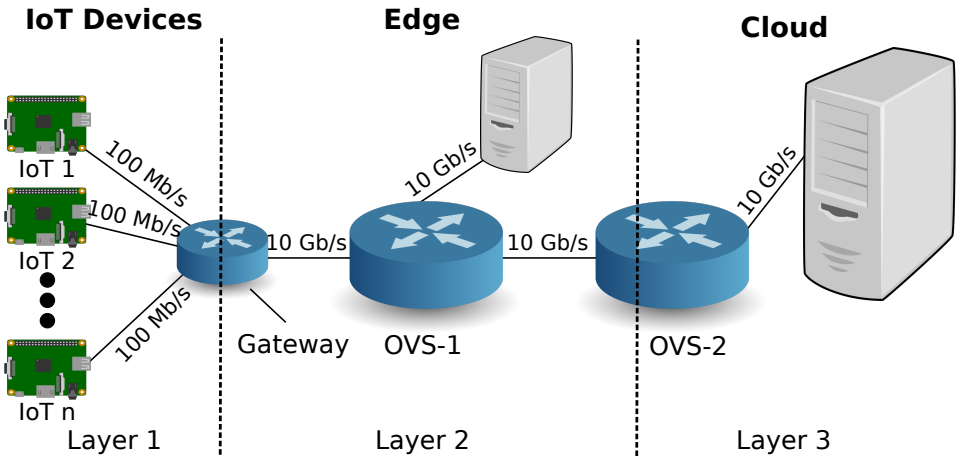


Figure 3.5: Experimental Cloud-Edge Infrastructure.

with an Intel® Xeon® E5-2620 at 2.10GHz and 64GB RAM, where the operators in the subsequence $Sink^{IoT}$ of the IoT layer will publish their output stream to be read by the subsequence $Source^{MD}$ on the MD layer.

The MD and cloud layers contain four servers with an Intel® Xeon® X5550 at 2.67GHz with 32GB RAM and a NetFPGA card with four 10Gb/s ports. Two of those servers are used for the operator deployment, one in each layer (MD and cloud), and the two remaining servers are used as routers for the layers, running instances of Open vSwitch[1] 2.9 (OVS). The servers and their respective OVS are connected by 10Gb/s links, but due to limitations of the NetFPGA driver and from OVS the maximum bandwidth achieved is ≃2.3Gb/s. The latency between the gateway and the MD resource is configured as ≃24ms and the latency between the MD resource and the cloud is ≃50ms [29].

Data Stream Processing Application

The proposed model estimates the throughput based on a deployment plan for a DSP application. In this experiment we use a sentiment analysis application [38], which evaluates the positive or negative sentiment associated with a tweet. The application structure is depicted in Figure 3.6. The *data source* of the application is a data set with 50K tweets as JSON-format files with sizes ranging from 4 to 24 KB. The tweet data set is recursively read during the application execution. The *data sink* stores the data produced by the last operator.

The application is composed of a pipeline with five operators. The selectivity for operators 1 to 5 are 9.5%, 0.4%, 8.7%, 52.4% and 117.1% respectively, and the transformation on the data stream applied by each operator are described as follows:

- **Language Filter** (*Op* 1): filters and discards every tweet that is not in English.
- **Special Characters Filter** (*Op* 2): removes non-letter characters from the tweet text.
- **Non-sentiment Words Filter** (*Op* 3): removes irrelevant or non-sentiment words (*e.g.* the, and, or).

[1] https://www.openvswitch.org/

- **Positive/Negative Words Counter** (*Op* 4): creates a score and counts the number of words with positive and negative sentiments.
- **Scorer** (*Op* 5): scores the sentiment of the tweet based on the number of positive and negative words.

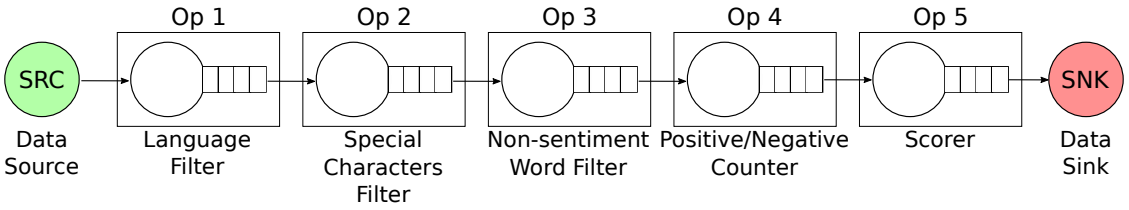


Figure 3.6: The operator graph for the sentiment analysis application.

Since to the best of our knowledge there is no framework that address the cloud-edge infrastructure and therefore attend our needs, we use multiple software solutions to ease de deployment on such infrastructure. As depicted in Figure 3.6 each operator is coupled with an output queue that works as a buffer for the output data stream that will be read by the following operator in a First In First Out (FIFO) manner. To implement the queues for each operator we resorted to a lightweight message broker that could be executed on the Raspberry Pi's, called Mosquitto MQTT [2]. The operators are implemented in Java using the DSP framework Apache Edgent [3]. Apache Edgent is lightweight framework solution to run on edge resources. Each operator is executed as an independent instance of the Apache Edgent framework.

Scenarios

We designed six scenarios to compare the estimated and the actual throughput (Table 3.2), each with a different deployment plan for the sentiment analysis application (Figure 3.6) considering the cloud-edge infrastructure (Figure 3.5). The deployment proposal for each scenario differs regarding the operator assigned for the sequence of each layer. Data source and data sink have fixed position at the IoT and cloud sequence respectively. Also, each sequence has at least one operator; therefore *Op* 1 is always in the IoT sequence, and *Op* 5 is always in the cloud sequence. The sequence of each layer is replicated for all devices of the layer.

The processing requirements of the operators change in each scenario. The requirements are defined after profiling the application execution according to the deployment plan for the scenario, where the processing capacities of the resources are shared among the operators deployed on the device. Since *Op* 1 reads the tweets from a data set, the rate at which it reads tweets is affected by the processing requirements of the operator, hence the generation rates change according to the scenario.

The evaluation executed the DSP application for 1440 seconds for each scenario. This time is large enough for the application to execute beyond the warm-up phase, a period needed for the application to achieve a state where all the operators are processing messages at a given time. From observation, the noise created by the warm up phase always lasts less than 300 seconds. Therefore, to ensure that we collect data during the stable phase, we disregard the first 300 seconds.

[2] https://mosquitto.org/
[3] http://edgent.apache.org/

Table 3.2: Deployment scenarios for throughput estimation.

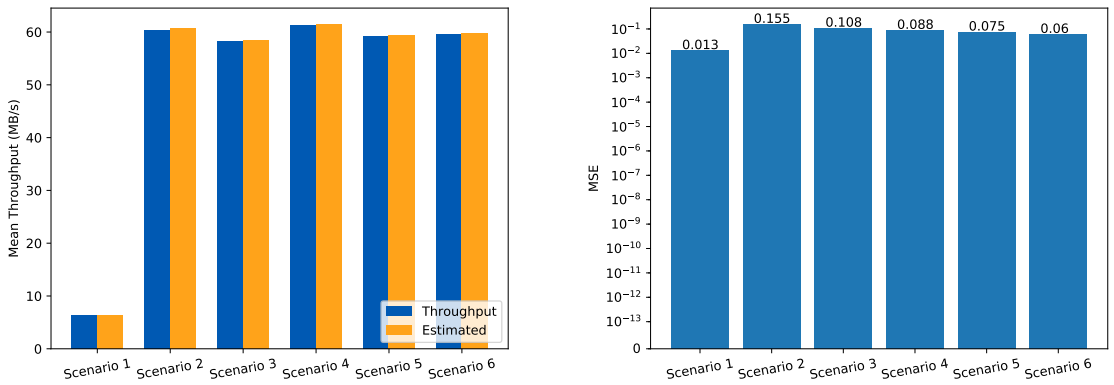
Scenario	IoT	MD	Cloud
1	*Op* 1 *Op* 2 *Op* 3	*Op* 4	*Op* 5
2	*Op* 1 *Op* 2	*Op* 3 *Op* 4	*Op* 5
3	*Op* 1	*Op* 2 *Op* 3 *Op* 4	*Op* 5
4	*Op* 1	*Op* 2 *Op* 3	*Op* 4 *Op* 5
5	*Op* 1	*Op* 2	*Op* 3 *Op* 4 *Op* 5
6	*Op* 1 *Op* 2	*Op* 3	*Op* 4 *Op* 5

3.4.3 Performance Evaluation Results

The average of actual and estimated throughput is depicted in Figure 3.7(a). Once the system reaches a stable state, the throughput does not vary much. The estimated throughput is close to the actual throughput. Even in the first scenario that produced a very low throughput, the model estimated a close value.

The reason for the low throughput under the first scenario is that the deployment plan deploys three operators in a device with constrained capacity (*i.e.* a Raspberry PI). The third operator in particular, that removes non-sentimental words from tweets, has a significant impact on the application performance. It is very demanding in terms of processing capacity. As it runs on a constrained device, it affects the throughput of the whole application. This reinforces the importance of setting the right requirements for each operator.

Another indicator of the quality of the proposed model is the error of less than 1% in the difference between the estimation and the collected values. We also computed the mean square error to highlight estimated outliers and verify the quality of results. The lower the values the better the quality of the estimation. As depicted in Figure 3.7(b), our estimation achieved low values reinforcing the precision on the proposed model.



(a) Mean throughput and estimation.

(b) Mean square error.

Figure 3.7: Results throughput estimation for DSP applications on cloud-edge computing.

3.5 Conclusion

This chapter presented an architecture designed to ease the deployment and management process of DSP application on the cloud-edge infrastructure. Beyond the proposed architecture this chapter presented a model to solve the operators placement problem, and a model to estimate the throughput of DSP application on cloud-edge infrastructures. The model to solve the operator placement is based on MILP, and comprise infrastructure and application models designed to deploy DSP applications on cloud-edge environments. While the infrastructure model takes into consideration the resource heterogeneity in terms of computational power and network latencies faced by resources, the application model contains characteristics previously overlooked in the literature, such as selectivity, data transformation pattern, probability of load distribution for the streams, and requirements obtained based on profiling information. The model splits an operator into multiple replicas, each responsible for processing a fraction of the operator's overall load, and the requirements are adapted according to the load of the replica and the device in which it should be deployed.

The objective of the proposed model is to reduce the application end-to-end latency and deployment costs. The application end-to-end latency is minimized though the Aggregate data Transfer Time (ATT), which aims to reduce the amount of data crossing the Internet, which faces network congestion. Regarding the computational costs, we built a model that considers both computational and network costs, using elements from solutions proposed by Amazon, such as AWS Fargate Pricing [71], AWS Direct Connection [72], AWS IoT Core [77] and AWS Private Links [78].

The scheduling model is used in subsequent chapters to devise a solution for placement and parallelism of DSP applications. This chapter also provided a throughput estimation model that used the scheduling model to compute the throughput that a given deployment plan can achieve, considering the challenges introduced by the cloud-edge environment and the operator replication process. The quality of estimation model is measured by how accurate they are when compared with the reality. and according to evaluation the estimation model proposed in this thesis has an accuracy of 99%.

Chapter 4

Optimal Scheduling Solution for Stream Processing Applications on Cloud-Edge Infrastructures

Contents

4.1 Introduction

Problems in placement and scheduling of Data Stream Processing (DSP) applications are often NP-Hard even when ignoring cloud-edge infrastructure [28]. The inclusion of cloud-edge infrastructure adds more challenges as a placement solution needs to consider the high degree of heterogeneity created by the combination of cloud and edge computing resources and the communication between operators deployed at different places of the infrastructure since data transfers often traverse Internet links. Moreover, edge resources are computationally constrained. Thus a solution needs to adapt the operators deployed on such resources to avoid infrastructure saturation and performance degradation.

In this chapter, we use the Mixed Integer Linear Programming (MILP) scheduling model proposed in Section 3.3 into a solution named Cloud-Edge data Stream Placement (CESP) to deploy and replicate operators throughout the cloud-edge infrastructure with bandwidth guarantees. This solution accounts for cloud and edge resources, where the edge infrastructure is composed of Internet of Things (IoT) resources and Micro Datacenters (MDs). MD and cloud resources incur some monetary costs for deployment, whereas IoT resources are free of charge.

IoT and MD resources are computationally constrained, but can provide low latency as they are often closer to data sources. CESP aims to explore the trade-off between paid and free of charge resources, computationally constrained and low latency resources, in order to minimize the application end-to-end latency and deployment costs, and ensure the maximum theoretical throughput that the application can achieve.

4.2 Impact of Three-Layered Cloud-Edge Infrastructure

As discussed beforehand, a cloud-edge infrastructure can comprise hundreds or even thousands of resources [79], organized into three layers (IoT, MD and cloud). Since IoT and cloud resources have clear benefits to DSP applications on IoT scenarios, where IoT provides low network latency and the cloud offers high computational power, one might wonder on the benefits that using MDs brings. A MD acts as a middle ground between IoT and cloud resources both in terms of network latency and computational power. Therefore, instead of evaluating only CESP considering the entire resource search space and compare it against a standard approach that is focused only on cloud deployment, we also investigated the the impact of using MDs to host operator replicas.

Hence, CESP is evaluated as two approaches, where each approach differs from the other regarding which part of the infrastructure is considered in the search space. The first approach, named CESP–All, considers all the infrastructure resources as possible candidates to host either an operator or one of its replicas. The second approach, called CESP–IC, investigates if MDs have any effect on the overall performance of the application and on its deployment costs. CESP–IC aims to place operator replicas in the low-end (IoT) resources as much as possible. Unlike CESP–All, CESP–IC is likely to generate many more replicas in order to meet the operator requirements by using low-end devices. We want to evaluate this increase in the number of replicas. The standard solution against which the model versions are compared is called Cloud-Only, which applies a random walk to place the application operators only onto cloud resources.

The placement solutions presented in this chapter therefore consist in solving the optimal scheduling model proposed in Section 3.3 for different sets of resources (*i.e.*, the entire infrastructure, or a subset of it). This approach is further improved in the next chapter, which presents an additional solution to reduce the search space during the optimization of placement and parallelism.

4.3 Performance Evaluation

CESP is a scheduling solution for DSP applications on IoT scenarios that considers cloud-edge infrastructures for deployment, and that explores operators replication and bandwidth guarantees to achieve performance improvement. Then, our evaluation is divided in two scenarios. In the first scenario we focus into investigate the benefits of bandwidth guarantees, where we compare CESP against the standard solution Cloud-Only with and without bandwidth guarantees. In the second scenario we evaluate CESP–All and CESP–IC and compare it against the standard approach Cloud-Only, focusing on the benefits of operator replication and the usage of cloud-edge infrastructure to the application end-to-end latency and deployments costs.

Beyond the discussion of the obtained results, the rest of this section presents the performance metrics evaluated, cost elements and the experimental setup. For the most part the experimental setup is equivalent on both scenarios of the performance evaluation, with the exception of the WAN bandwidth considered. Before the discussion of the results of each evaluation

the considered WAN bandwidth is explained.

4.3.1 Experimental Setup

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

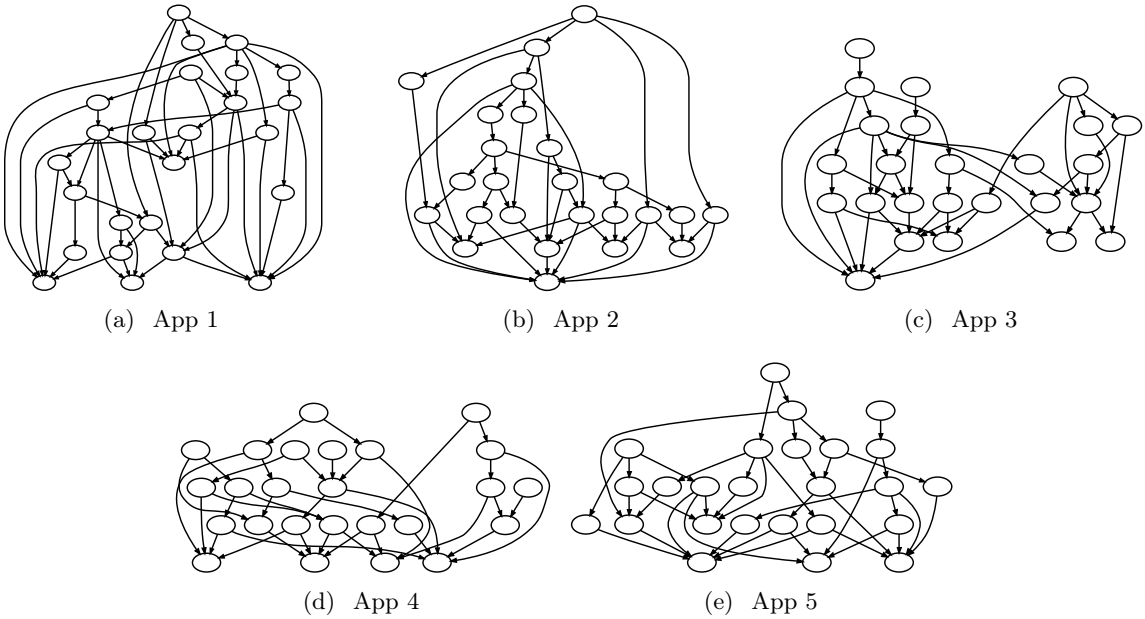
Resources within a site communicate via a LAN, whereas IoT sites, MDs, and cloud are interconnected by a single WAN path. The LAN has a bandwidth of 100 Mbps and 0.8 ms of latency. The WAN bandwidth is shared on the path from the IoT to the MD or to the cloud and its capacity is defined in each scenario, and the latency from IoT is 20 ms and 90 ms to the MD and cloud, respectively. The latency values are based on those obtained by empirical experiments carried out by Hu *et al.* [29].

To evaluate CESP considering diverse and generic applications, we crafted multiple application graphs with various shapes and sizes. Existing work evaluated application graphs of several orders and interconnection probabilities, usually assessing up to 3 different graphs [11, 57, 62, 70]. Using a built-in-house python library, we built five graphs to mimic the behavior of large DSP applications. The graphs have various shapes and data replication factors for each operator, as depicted in Figure 4.1. The applications have 25 operators, often more than what is considered in the literature [80]. They also have multiple sources, sinks, and paths, similar to previous work by *Liu and Buyya* [70]. As the present work focuses on IoT scenarios, the sources are placed on IoT resources, and sinks are uniformly and randomly distributed across layers as they can be acting as actuators – except for one sink responsible for data storage, which is placed in the cloud.

The operator properties are based on the RIoTBench [39]; an IoT application benchmark that offers 27 operators common to IoT applications and 4 datasets with IoT data. The experiments use the CITY dataset with 380 bytes messages collected every 12 seconds containing environmental information (temperature, humidity, air quality) from 7 cities across 3 continents. It has a peak rate of 5000 tuples/s, which is continuous and divided among sources. The remaining properties are drawn from the values in Table 4.1.

The Req^{j}_{cpu}[1] of an operator j can be computed based on measurements obtained via application profiling, including Ref^{j}_{cpu} and Ref^{j}_{data}, using techniques proposed in existing work [76]. In practice, the arrival byte rate AR^{j}, the probability that an upstream operator i sends data to j, *i.e.* $\rho^{i \to j}$, selectivity S^{j}, and data transformation pattern C^{j} could be average values obtained via application profiling. However, to create a worst-case scenario in terms of load, $\rho^{i \to j}$ is set to 1 for all streams in each application request, meaning that operator j replicates

[1]For a list of the notation used in this chapter, please refer to Table 3.1.



(a) App 1 (b) App 2 (c) App 3

(d) App 4 (e) App 5

Figure 4.1: Application graphs used in the evaluation.

Table 4.1: Operator properties in the application graphs.

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

its outgoing messages to all its down streams. As CESP creates multiple replicas, $\rho^{i \to j}$ gets divided among instances of operator j, hence creating variations on the arrival rate of downstream operators during runtime. The operator processing requirements estimated by the model may not be enough to handle the actual load during certain periods, so resulting in large operator queues. To circumvent this issue, we add a small safety margin, the β factor, which is a percentage increase in the application requirements estimated by CESP. A β too high results in expensive over-provisioning. We evaluated multiple values of β and set it to 10%, which gives a performance boost to handle the queues without incurring high costs.

4.3.2 Price Model

The price for resources is derived from Amazon AWS services, considering the US East Virginia location. The CPU and memory prices are computed based on the AWS Fargate Pricing [71] under a 24/7 execution. We consider a Direct Connection [72] for bandwidth guarantees between the IoT site and the AWS infrastructure. As DSP applications generate large amounts of data, we

Table 4.2: Computing and network costs.

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

consider a Direct Connection of 10 GB/s. The data sent from IoT sites to AWS infrastructure uses AWS IoT Core [77]. Connections between operators either on MD or IoT resources to the cloud use Private Links [78] for bandwidth guarantees. Amazon provides the values for CPU, memory, and network as, respectively, a fraction of a vCPU, GB, and Gbps, but in our formulation, the values for the same metrics are computed in CPU units ($100 \times num\_cores$), bytes and Mbps. The values provided by Amazon, but converted to the scale used in our formulation are presented in Table 4.2. As the environment combines both public and private infrastructure, deployment costs are applied only to MDs and cloud resources, the network between these two, and the network between these two and IoT resources. The communication between IoT resources is free since they are on the same private network infrastructure.

4.3.3 Evaluated approaches and metrics

Five different configurations of deployment requests are submitted for each application during 120 simulated seconds. The reported values for each application are averages of these five executions. Each request has a different placement for sources and sinks, but still respecting the rule of sources always on IoT resources and at least one sink in the cloud. The operator properties such as selectivity and data transformation factor vary across configurations.

As for performance metrics, we consider:

- *throughput*, which is the processing rate, in bytes/s, of all sinks in the application; and
- *end-to-end latency*, which is the average time span between the generation until the message reaches a sink.

CESP takes the throughput into account in the constraints and the end-to-end latency indirectly by optimizing the Aggregate data Transfer Time (ATT).

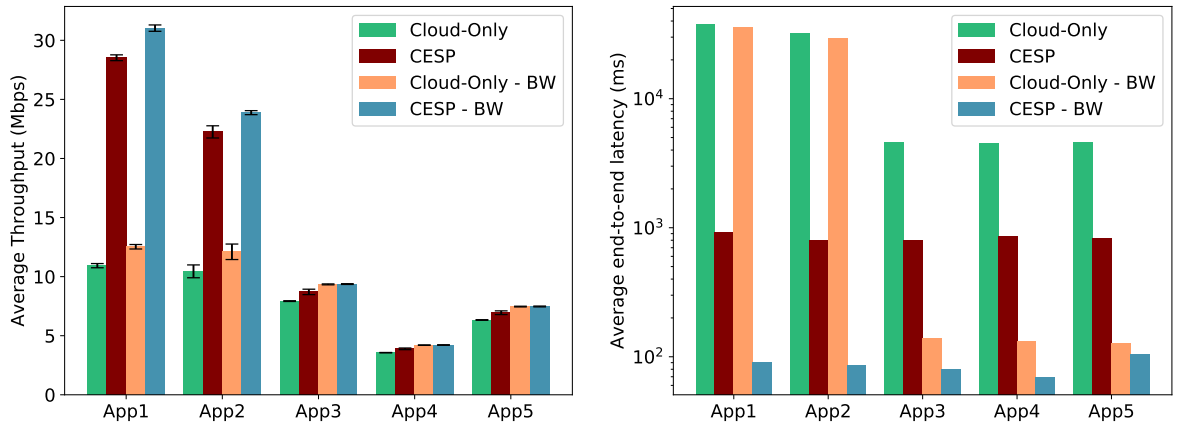
4.3.4 No Bandwidth Control versus Bandwidth Control

In this scenario we focus on evaluating how the bandwidth control affects the overall performance of an application. To create a background traffic to resemble real traffic on the Internet we rely on studies in the literature that show that around 80% of the Infrastructure Service Provider (ISP) infrastructure is used at any given time [81, 82]. We mimic this characteristic to the WAN links of the simulated infrastructure, by creating multiple resources that continuously send messages traversing the Internet links, to create a load of around 80% of the capacity of such links.

The number of resources to produce background traffic is determined by the bandwidth on WAN links and the size of messages created by these resources. Regarding the size of messages produced we once again rely on existing work in the literature. The Internet is a very heterogeneous environment, which makes it difficult to determine an average, or even a range for the size of messages. However a study developed by Hu *et al.* [29] that evaluated messages produced at the edges of the network, text messages had around 10 bytes, pictures/objects 50 KB, and voice records around 200 KB. The study developed by Hu *et al.* proposed a discrete set of values, but to create a more heterogeneous environment, we used a continuous set of values ranging between 10 bytes and 200 KB messages.

With respect to the bandwidth of WAN links, we set it to 1Gbps. Other values higher than this would result into a simulated environment overloaded with a tremendous amount of messages produced as background, which would increase the time to produce results. Besides, as the number of messages produced increases, the memory requirements to run the simulation increase as well, which at some point exceeds the physical memory available in the machine running the simulation. With 1Gbps WAN links and messages with sizes ranging between 10 bytes and 200 KB, to keep an average of 80% usage of WAN links, we need 500 resources to produce background traffic.

We compared CESP against the standard Cloud-Only solution with and without bandwidth control. Hereafter, CESP and Cloud-Only with bandwidth control are called CESP – BW and Cloud-Only – BW, respectively. To impose bandwidth guarantees we consider services such AWS Direct Connection [72] for WAN links and AWS Private Links [78] for LAN link inside the MD and cloud infrastructures. Figure 4.2 summarizes results for throughput and application end-to-end latency. The first thing to notice is that in some scenarios CESP overcomes the performance of Cloud-Only – BW. This means that even for application with data produced at the edges of the network, scheduling decisions might have a bigger impact on the overall performance than bandwidth control, both on throughput and application end-to-end latency. When we compare CESP against CESP – BW and Cloud-Only against Cloud-Only – BW, it is easy to identify that bandwidth guarantees can boost the performance of DSP applications on IoT scenarios.



(a) Throughput

(b) End-to-end latency

Figure 4.2: Throughput and end-to-end latency under Cloud-Only and CESP with and without bandwidth control.

However, there are scenarios where CESP overcomes Cloud-Only – BW both on throughput and end-to-end latency and others where it does not happen. The reason for that is the structure of the applications. CESP overcomes Cloud-Only – BW for applications App1 and App2. If we look at Figure 4.1 we can observe that App1 and App2 can be considered as "thin" applications, and Apps 3 to 5 "fat" applications. While in fat applications there is a lot of communication between operators due to the high degree of data replication of each operator, in thin applications there are just a few operators that send data to more than one operator. This has a bigger effect on the overall network requirements and susceptibility to network congestion interference.

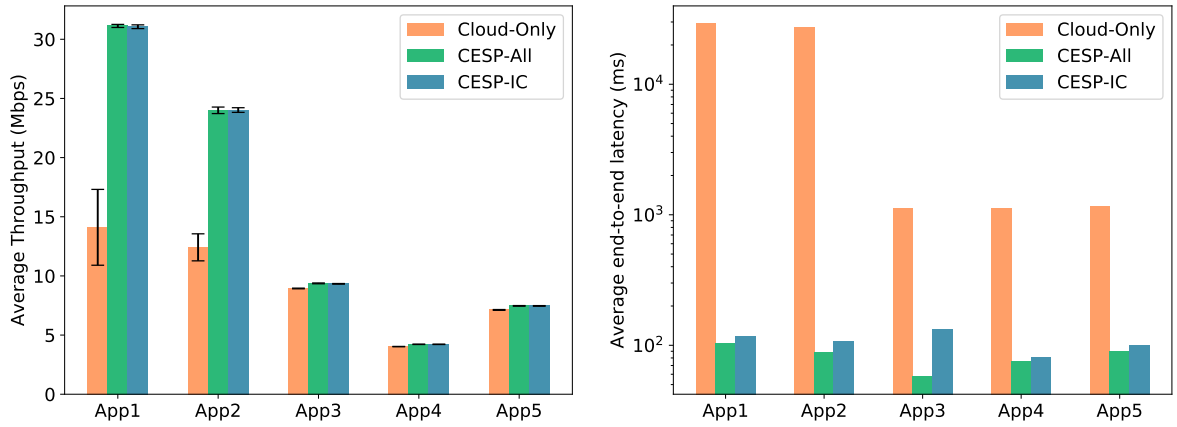
Hence, with less network requirements, bandwidth control does not play a key role in the overall performance of DSP applications on IoT scenarios. Since App1 and App2 do not imposed heavy network requirements, there is more room to improve it regarding scheduling decisions. Contrary to the bandwidth control, scheduling decisions have a major impact in the overall performance of DSP applications. Since CESP does not optimize throughput, Figure 4.2(a) does not depict how good CESP can perform, but how poorly Cloud-Only perform and how much some applications can suffer from such deployment. On the the other hand, the application end-to-end latency is a performance metric that CESP optimizes, the it show improvements on any kind of application, and that the bandwidth control can be leverage to improve even further the performance of some applications.

4.3.5 CESP versus the Standard Approach

In this scenario we consider bandwidth guarantees to all evaluated approaches. The network can hence provide higher bandwidth availability, and to this end the WAN bandwidth considered is of 10 Gbps. Throughput results are summarized in Figure 4.3(a). Under most scenarios, Cloud-Only and CESP achieve similar throughput. However, under App1 and App2, Cloud-Only performs much worse than both CESP versions because these apps have less data sources, then, with a single source producing messages every few milliseconds that goes from the edges of the network to the cloud, takes more time in the network then processing. That becomes an issue to Cloud-Only, while both CESP versions are able to cope with this scenario better. Apps 3-5, on the other hand, have more data sources spread across many resources thus producing more data simultaneously, increasing the amount of data reaching the cloud, then Cloud-Only can handle similar to CESP.

Processing data only in the cloud has a negative effect on end-to-end latency, as shown in Figure 4.3(b). The network becomes a bottleneck, especially in the LAN sections. Messages are queued, producing a high end-to-end latency for Cloud-Only, even in scenarios where Cloud-Only had similar throughput. CESP tackles this network problem by placing communicating operators closer to one another in terms of network latency – *i.e.*, placing sources and their immediate downstream operators on the same resource. As IoT resources are computationally constrained, placing communicating operators on the same device becomes challenging. CESP breaks an operator into small replicas, thus allowing this co-placement. Even if the replica processes only part of an operator's load, it helps by reducing the data sent through the network, hence reducing congestion. With this process of co-placement and operator replication, CESP reduces the end-to-end latency by at least 80%

Table 4.3 contains the average CPU and memory requirements per operator instance for the evaluated DSP applications. As Cloud-Only does not create replicas, its reported values are the average requirements per operator, computed as the one used by both CESP implementations. Results demonstrate that CESP divides the operator into multiple replicas, each of which has



(a) Throughput

(b) End-to-end latency

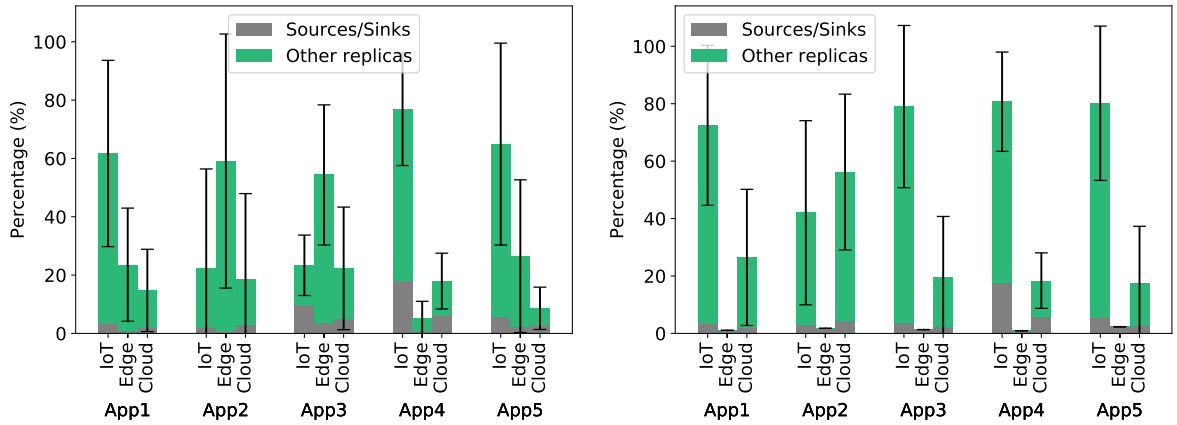
Figure 4.3: Throughput and end-to-end latency under Cloud-Only and CESP.

Table 4.3: Average resource consumption per operator instance.

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

significantly smaller requirements allowing for better utilization of IoT and MD resources with co-placement at the edges of the network hence experiencing lower network latency. By breaking an operator into replicas with lower requirements and using IoT and MD resources, CESP achieves better end-to-end latency and deployment costs.

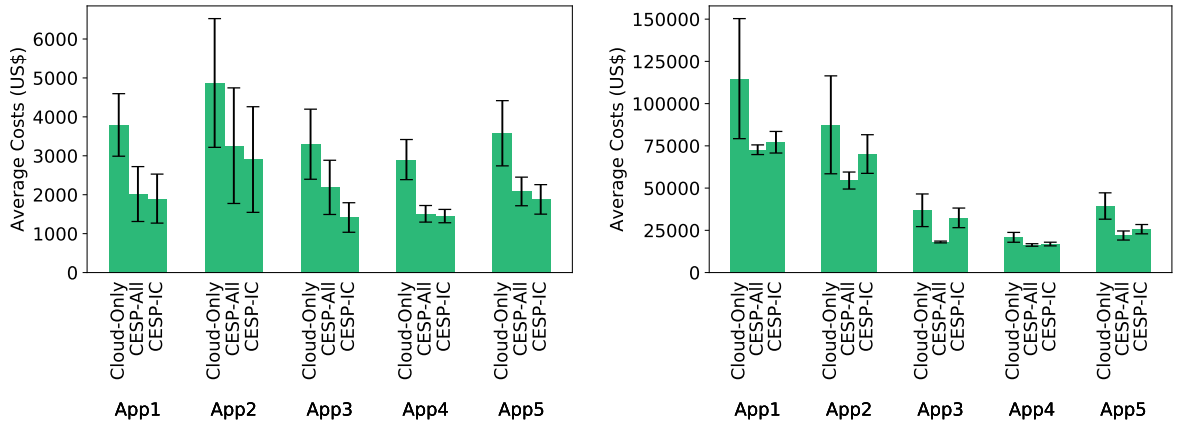
Figure 4.4 shows the percentage of replicas deployed in each layer. The bottom part of each bar presents the percentage of sources and sinks deployed in the respective layer, whereas the top part corresponds to other operators. CESP–All provides better end-to-end latency than CESP–IC due to the use of MD resources. MD resources are computationally more powerful than IoT resources, so enabling the co-placement of more replicas on the same node. When the volume of data processed by each operator grows, it becomes inefficient to create several replicas on IoT resources. CESP creates multiple replicas of communicating operators as pipelines and places each pipeline in a different device, resulting in the end-to-end latency improvement of CESP–All when compared with CESP–IC. As CESP–IC does not use MD resources, it continues to explore IoT resources, by creating multiple replicas combined into pipelines into different devices. The downside of using IoT resources for this is that the pipeline is shorter, requiring the use of the network to communicate with the downstream replica, resulting in higher end-to-end latency.



(a) CESP–All (b) CESP–IC

Figure 4.4: Replica distribution per resource for both CESP versions.

Figure 4.5 shows the deployment costs. Cloud-Only does not consider IoT resources, which are free of charge, and does not reduce the amount of data traversing the Internet. It yields the highest deployment cost, both computational and network. Along with the end-to-end latency gain from co-placing small replicas into IoT resources, CESP–IC explores such devices as they are free of charge. CESP–All, which explores MDs in addition to IoT resources, is able to deploy more replicas at the edges of the network, thus reducing the network usage and costs. CESP–All experiences the cheaper deployment costs.



(a) Computational costs (b) Network costs

Figure 4.5: Computational and network costs under Cloud-Only, CESP–All and CESP–IC.

4.4 Conclusion

This chapter presented CESP, a solution for the operator placement and parallelism of DSP applications that uses the MILP scheduling model proposed in Section 3.3 to optimize the

application end-to-end latency and deployment costs. CESP combines profiling information with the computed amount of data that each operator should process in order to obtain the application processing requirements so that each operator can handle the arriving load. CESP also creates multiple lightweight replicas to offload operators from the cloud to the edges of the network hence obtaining lower end-to-end latency. Since the communication between IoT, MD and cloud layers crosses the Internet, to reduce the network interference CESP imposes bandwidth guarantees thought services such as AWS Private Links [78] and AWS Direct Connect [72].

We evaluated CESP under two scenarios. Firstly, CESP was evaluated to identify the effects and benefits of imposing bandwidth guarantees, and was compared against the standard solution Cloud-Only, where both solutions were evaluated with and without bandwidth guarantees. Results show that the bandwidth guarantees for DSP applications deployed on IoT scenarios can boost the performance of the application. But its benefits are limited to characteristics of the application, where on some cases scheduling decisions have a bigger performance impact.

Secondly, two versions of CESP were evaluated using various applications with different configurations in terms of selectivity, data transformation pattern, and CPU and memory requirements. Both versions provide at least $\simeq 1\%$ throughput improvement, $\simeq 80\%$ end-to-end latency reduction and deployment costs $\simeq 30\%$ cheaper than a traditional placement scheme. The results also show that by using MD resources, the end-to-end latency can be improved by at least $\simeq 6\%$ and deployment costs reduced by $\simeq 4\%$.

Chapter 5

Pruning Heuristics for Scheduling Stream Processing Applications on Large Cloud-Edge Infrastructures

Contents

5.1 Introduction

A cloud-edge infrastructure can contain numerous geographically distributed resources. A solution to the operator placement problem that addresses cloud-edge infrastructure requires scalability, which is not something that optimal solutions usually can provide.

In Chapter 4 we have shown that Cloud-Edge data Stream Placement (CESP) is able to overcome the standard solution, Cloud-Only, both on performance metrics and costs. In this chapter we intent to make CESP more scalable by proposing Cloud-Edge data Stream Placement with Resource Selection (CESP–RS). CESP–RS combines CESP with a resource selection technique that aims to reduce the search space by sorting all the resources in the infrastructure and selecting a resource subset based on a worst-fit technique.

5.2 Resource Selection Technique

The three-layered cloud-edge infrastructure may contain thousands of computing resources resulting in an enormous combinatorial search space when finding an optimal operator placement. This work therefore proposes a pruning technique that reduces the number of evaluated resources and finds a sub-optimal solution under feasible time. The proposed solution extends the *worst-fit* sorting heuristic from Taneja *et al.* [61] by applying a resource selection technique to reduce the number of considered computing resources when deploying operators. The proposed solution selects from each layer a subset of $\mathcal{R}$[1] resources that are candidates to deploy replicas of the application.

Since in this thesis we consider the Internet of Things (IoT) infrastructure as private, and the data is being generated at this infrastructure, its location needs to be provided in the deployment request. Each deployment request can consider one or multiple IoT infrastructures (*iotSite*) that serve as input to the resource selection technique, depicted in Algorithm 1. For each *iotSite* received as input it selects one Micro Datacenter (MD)(*mdSite*), the one with the shortest latency. Also, all the MDs that should contain a data sink are selected (line 2). For last, one cloud site is chosen based on the average latency-closeness to the selected *mdSite* as well as the position of data sinks (line 3). The information about data sinks is also provided at the deployment request. The rationale behind the selection of sites for each layer is that it reduces the search space based on closeness, to resources that are more likely to provide performance gain to Data Stream Processing (DSP) applications. Then, for each layer a subset of resources is selected using the *GetResources* function.

The *GetResources* function has as input the layer name, the vector of selected sites in the layer and the set of operators. First, it calls *GetResourcesOnSites*, to get all computing resources from the selected sites (line 12). Second, it selects resources that host sources or sinks (lines 13-17). Third, CPU and memory requirements from the operators that are neither sources or sinks are summed to $ReqCPU$ and $ReqMem$, respectively (line 20). When the evaluated layer is IoT, the set of resources is sorted by CPU and memory capacity in descending order and $ReqCPU$ and $ReqMem$ are used to select a subset of computing resources whose combined capacity meets the requirements (lines 29-34). For the other two layers, the function iterates through the list of operators selecting a worst-fit resource that supports the operator's requirements. Since the goal is just to select candidate resources and not a deployment placement, if there is no resource fit, it ignores the operator and moves to the next one (lines 23-27). Resources that are selected as worst-fit for one operator cannot be selected for another operator At last, the combination of resources evaluated by the model contains those selected in each layer.

5.3 Performance Evaluation

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

5.3.1 Experimental Setup

We perform an evaluation in two steps as follows. First CESP is compared against CESP–RS to evaluate the effects that the resource selection has on the quality of the placement solutions

[1]For a list of the notation used in this chapter, please refer to Table 3.1.

Algorithm 1: Resource selection technique.

```
1  Function ResourceSelection(iotSite, O)
2      mdSite ← GetEdgeSite (iotSite)
3      cloudSite ← GetCloudSite (edgeSite)
4      Selected ← GetResources (IoT, iotSite, O)
5      Selected ← Selected ∪ GetResources (MD, GetEdgeSite (iotSite), O)
6      Selected ← Selected ∪ GetResources (cloud, GetCloudSite (iotSite), O)
7      return Selected
8  Function GetResources(resourceType, site, O)
9      Selected ← {}
10     ReqCPU ← 0
11     ReqMem ← 0
12     Resources ← GetResourcesOnSite (site)
13     foreach j ∈ (Source^O ∪ Sink^O) do
14         if j.placement ∈ Resources then
15             r ← j.placement
16             Selected ← Selected ∪ r
17             Resources ← Resources − r
18     foreach j ∈ (O − (Source^O ∪ Sink^O)) do
19         ReqCPU ← ReqCPU + j.CPU
20         ReqMem ← ReqMem + j.Mem
21         if resourceType! = IoT then
22             Sort (Resources)
23             foreach r ∈ Resources do
24                 if r.CPU ≥ j.CPU and r.Memory ≥ j.Memory then
25                     selected ← selected ∪ r
26                     Resources ← Resources − r
27                     break
28     if resourceType == IoT then
29         Sort (Resources)
30         foreach r ∈ Resources do
31             if r.CPU ≤ ReqCPU and r.Memory ≥ ReqMem then
32                 Selected ← Selected ∪ r
33                 ReqCPU ← ReqCPU − r.CPU
34                 ReqMem ← ReqMem − r.Memory
35     return Selected
```

and on resolution time. Second, we compare CESP–RS against state-of-the-art approaches. The evaluations differ in the number of resources in the infrastructure and the solutions evaluated. Similar to the previous chapter, both evaluations are performed via discrete-event simulation using a framework built on OMNET++ to model and simulate DSP applications. We resort to simulation for this evaluation because it offers a controllable and repeatable environment. The model is solved using CPLEX v12.9.0.

The infrastructure comprises three layers with an IoT site, one MD and one cloud. The resource capacity was modeled according to the characteristics of the layer in which a resource is located, and intrinsic characteristics of DSP applications. IoT resources are modeled as Raspberry Pi's 3 (*i.e.,* 1 GB of RAM, 4 CPU cores at 1,2 GHz). As DSP applications are often CPU and memory intensive, the selected MD and cloud resources should be optimized for such cases. The offerings for MD infrastructure are still fairly recent and, although there is a lack of consensus surrounding what the MD is composed of, existing work highlights that the options are more limited than those of the cloud, with more general-purpose resources. In an attempt to use resources similar to those available on Amazon EC2, MD resources are modeled as general purpose t2.2xlarge machines (*i.e.,* 32 GB of RAM, 8 CPU cores at 3.0 GHz), and cloud servers are high-performance C5.metal machines (*i.e.,* 192 GB of RAM, 96 CPU cores at 3.6 GHz). Resources within a site communicate via a LAN, whereas IoTs, MDs, and cloud are interconnected by single WAN path. The LAN has a bandwidth of 100 Mbps and 0.8 ms latency. The WAN bandwidth is 10 Gbps and is shared on the path from the IoT to the MD or to the cloud, and the latency from IoT is 20 ms and 90 ms to the MD and cloud, respectively. The latency values are based on those obtained by empirical experiments carried out by Hu *et al.* [29].

Existing work evaluated application graphs of several orders and interconnection probabilities, usually assessing up to 3 different graphs [11, 57, 62, 70]. To evaluate CESP and CESP–RS we considered the five graphs described in the previous chapter, crafted to mimic the behaviour of large DSP applications using a built-in-house python library. The graphs are depicted in Figure 4.1 in Chapter 4. The applications have 25 operators, often more than what is considered in the literature [80]. They also have multiple sources, sinks and paths, similar to previous work by *Liu and Buyya* [70]. As the present work focuses on IoT scenarios, the sources are placed on IoT resources, and sinks are uniformly and randomly distributed across layers as they can be actuators – except for one sink responsible for data storage, which is placed on the cloud.

The operator properties were based on the RIoTBench IoT application benchmark [39]. RIoTBench offers 27 operators common to IoT applications and 4 datasets with IoT data. The CITY dataset is used with 380 byte messages collected every 12 seconds containing environmental information (temperature, humidity, air quality) from 7 cities across 3 continents. It has a peak rate of 5000 tuples/s, which in the experiments is continuous and divided among sources. The remaining properties are drawn from the values in Table 5.1.

We consider that Ref^j_{cpu}, Ref^j_{data}, the arrival byte rate AR^j, probability that an upstream operator i sends data to j $\rho^{i \to j}$, selectivity S^j, and data transformation pattern C^j, are average values obtained via application profiling, using techniques proposed in existing work [76]. With Ref^j_{cpu} and Ref^j_{data} we are able to compute requirements for each operator To create a worst case scenario in terms of load, $\rho^{i \to j}$ is set to 1 for all streams in the application request. As the model creates multiple replicas, $\rho^{i \to j}$ gets divided among instances of operator j, hence creating variations on the arrival rate of downstream operators during runtime. The operator processing requirements estimated by the model may not be enough to handle the actual load during certain periods, so resulting in large operator queues. To circumvent this issue we add a

Table 5.1: Operator properties in the application graphs.

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

Table 5.2: Computing and network costs.

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

small safety margin, the β factor, which is a percentage increase in the application requirements estimated by the proposed model. A β too high results in expensive over-provisioning. After multiple empirical evaluations, β was set to 10% of each replica requirement.

5.3.2 Price model

The price of using resources is derived from Amazon AWS services, considering the US East Virginia location. The CPU and memory prices are computed based on the AWS Fargate Pricing [71] under a 24/7 execution. Regarding the network, we consider a Direct Connection [72] between the IoT site and the AWS infrastructure. Direct Connections are offered under two options, 1 GB/s and 10 GB/s. As DSP applications generate large amounts of data, we consider the 10 GB/s offer. The data sent from the IoT to AWS infrastructure uses AWS IoT Core [77]. Connections between operators on the edge or on IoT resources to the cloud use Private Links [78]. Amazon provides the values for CPU, memory and network as, respectively, fraction of a vCPU, GB and Gbps, but in our formulation the values for the same metrics are computed in CPU units ($100 \times num\_cores$), bytes and Mbps. The values provided by Amazon converted to the scale used in our formulation are presented in Table 5.2. As the environment combines both public and private infrastructure, deployment costs are applied only to MD and cloud resources, the network between these two, and the network between these two and IoT resources. As IoT resources are on the same private network infrastructure, the communication between IoT resources is free.

5.3.3 Evaluated approaches and metrics

Five different configurations of deployment requests are submitted for each application. The reported values for each application are averages of these five executions. Each deployment request has a different placement for sources and sinks with sources always on IoT resources and at least one sink in the cloud. The operator properties such as selectivity and data transformation pattern vary across configurations.

As discussed earlier, the performance of DSP applications is usually measured considering two main metrics:

- *throughput*, which is the processing rate, in bytes/s, of all sinks in the application; and
- *end-to-end latency*, which is the average time span from when a message is generated until it reaches a sink.

The Mixed Integer Linear Programming (MILP) model takes the throughput into account in the constraints, and the end-to-end latency indirectly by optimizing the Aggregate data Transfer Time.

5.3.4 Resolution Time versus Solution Quality

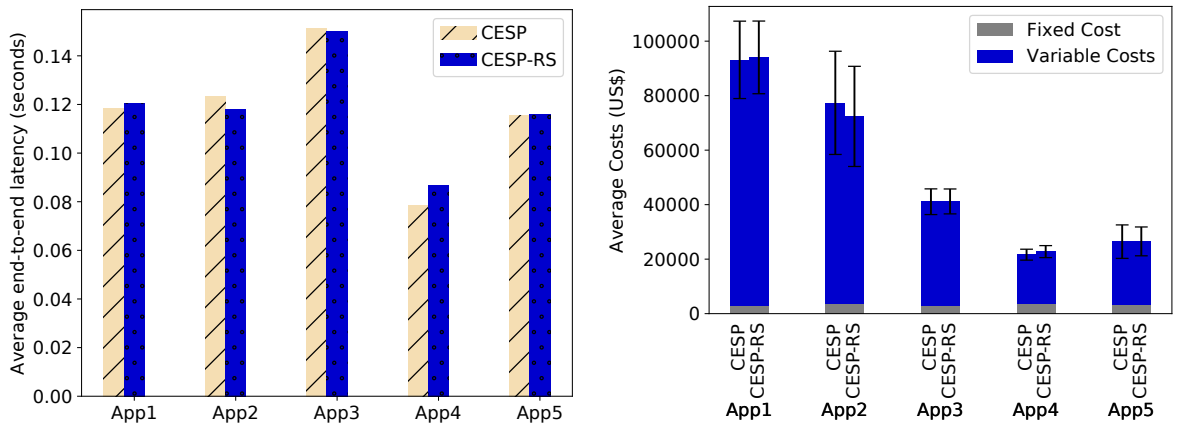
Here we evaluate how much the quality of a solution is sacrificed by reducing the search space. The simulation, which runs for 220 seconds, considers 100 IoT devices, a MD with 50 resources and a cloud with 50 resources. The throughput is the same in all scenarios since it is guaranteed as a model constraint.

Figure 5.1 shows the end-to-end latency and deployment costs under CESP and CESP–RS. There are some variations regarding the end-to-end latency both on CESP and CESP–RS. Since CESP–RS aims to reduce the search space, it might be counter intuitive to see cases where the resource selection with less options obtains better end-to-end latency, such as in App3. However, the objective function considers both latency and deployment costs as optimization metrics. As CESP searches to strike a balance between cost and end-to-end latency, the average deployment costs obtained with CESP–RS for App 3 (Figure 4.5) are higher. This behavior happens because under the limited search space, CESP–RS finds sub-optimal solutions, where the best trade-off resulted in better end-to-end latency. To do so, it needed to use more edge or cloud devices, which incurs higher computational and network costs.

As CESP considers the whole search space, it explores more options and yields better results. Despite reduced search space CESP–RS can produce very similar results – in the worst case yielding an end-to-end latency $\simeq 12\%$ worse, and deployment costs $\simeq 12\%$ higher. The resolution time (Figure 5.2), clearly shows that CESP considering the whole infrastructure faces scalability issues. Despite producing results that sometimes are worse than those achieved under CESP, CESP–RS can obtain a solution up to $\simeq 94\%$ faster. CESP–RS would yield even more similar results on a larger infrastructure because their search space is limited by the application size and requirements rather then by the infrastructure size.

5.3.5 CESP–RS versus the State-of-the-Art

CESP–RS is compared against two state-of-the-art approaches, namely *Cloud-Only* and *Taneja's Cloud-Edge Placement (TCEP)*. Cloud-Only applies a random walk considering only cloud resources, and TCEP is the work proposed by Taneja *et al.* [61], where all resources (IoT,



(a) End-to-end latency

(b) Deployment costs

Figure 5.1: End-to-end latency and deployment costs under CESP and CESP–RS.

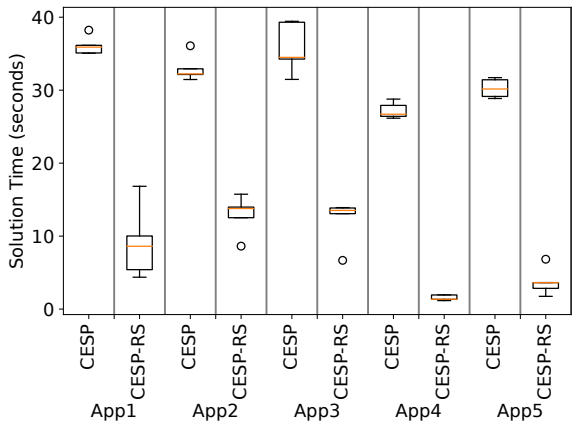
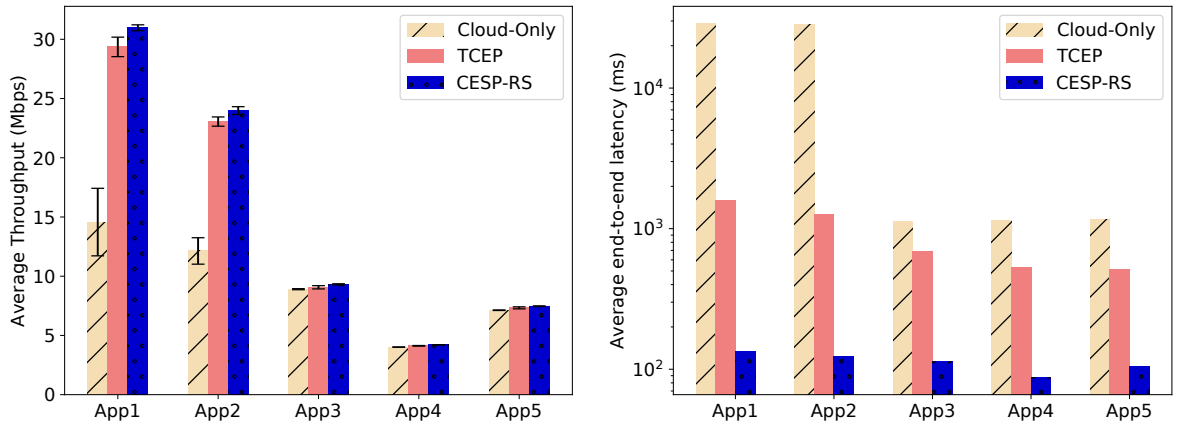


Figure 5.2: Resolution time to obtain a deployment solution.

MD and cloud) are sorted accordingly with their capacities, and for each operator it s elects a resource from the middle of the sorted list. This experiment was executed during 120 seconds and considered 400 IoT resources, 100 MD resources, and 100 resources on the cloud.

Figure 5.3 shows the throughput and end-to-end latency for all solutions, with averages for each application. Since CESP–RS guarantees a maximum throughput through a constraint, on the best case the other approaches would achieve the same values, and this can be observed on App3, App4 and App5. But under App1 and App2 Cloud-Only struggles because these are applications with a single data source each, then sending a single data message every few milliseconds from the edge all the way to the cloud, considering the network latency is inefficient, because messages spend more time in the network then processing. It is even more evident when we look at the application end-to-end latency of App1 and App2, where despite Apps 3 to 5 have a higher communication between operators, replicate more messages and have more data sources, they have a better application end-to-end latency on Cloud-Only. Because with more data sources, there are more messages crossing the internet simultaneously, then there is more data to process, and with more messages the average time between generating the message

and reaching a sink is smaller. When compared to Cloud-Only, TCEP provided better results, but still $\simeq$ 80% worse than the results provided by CESP–RS. CESP–RS achieves low values because, different from Cloud-Only and TCEP, it creates several replicas, being able to better explore the IoT resources considering their computational capacities and even further reducing the amount of data that is send through the internet, facing less network congestion.



(a) Throughput

(b) End-to-end latency

Figure 5.3: Throughput and latency under CESP–RS and state-of-the-art solutions.

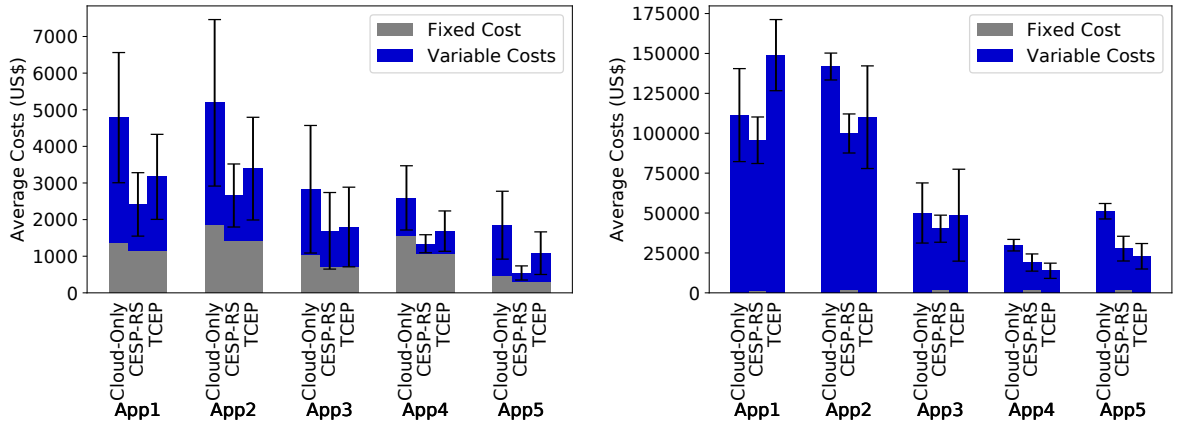
Figure 5.4 contains the costs results. Beyond better end-to-end latency, CESP–RS provides better computational costs. The reason that makes CESP–RS achieve computational costs at least $\simeq$ 6% better than the traditional approaches is the creation of replicas. The considered cost model, accounts for an IoT infrastructure without deployment costs, making such devices very attractive for deployment. Since IoT devices have constrained computational capacity, it is hard to deploy on such devices. Due to CESP, CESP–RS breaks an operator into several small replicas, allowing the use of IoT resources.

Regarding network costs, CESP–RS provides cheaper deployments on most cases except on App4 and App5. In these two applications, IoT resources support the operators' requirements without creating operator replicas allowing TCEP to exploit it and result in fewer data transfers. TCEP has higher computational costs because it cannot split operators into multiple replicas, thus resulting in placing the whole operator on powerful and expensive computing resources located on the cloud or a MD. When CESP–RS is compared to TCEP, it achieves a lower computational cost and a shorter end-to-end latency.

5.4 Conclusion

CESP explores profiling information on a reference infrastructure to obtain requirements for each operator. The profiling information combined with a stable arrival rate provides information on the maximum throughput that the application can achieve. Then, CESP enforces this maximum throughput and reduces the application end-to-end latency and deployment costs by creating multiple lightweight replicas to offload them from the cloud to edge resources.

Optimal solutions, such as CESP, are known for scalability issues, and the cloud-edge infrastructure is composed of numerous geographically distributed resources. To overcome the



(a) Computational costs

(b) Network costs

Figure 5.4: Computational and network costs under CESP–RS and state-of-the-art solutions.

scalability issue faced by CESP, this chapter presented a resource selection technique that reduces the number of resources evaluated during placement and parallelization decisions. The proposed model, coupled with the resource selection technique (*i.e.*, CESP–RS), is 94% faster than solving CESP alone, produces solutions that are only 12% worse than those achieved under CESP, and performs better than traditional and state-of-the-art approaches.

Chapter 6

Conclusion and Future Directions

Contents

6.1 Discussion and Contributions

The most diverse areas of society are permeated by connected systems [1]. Such connected systems are generating ever growing amounts of data that are used on decision making. This is part of the data driven economy we are currently living, where organizations and individuals take decisions upon information obtained over collected data. In order for the data to become information it requires some processing [3]. A common approach to process data to obtain information is through *batching processing* where a large amount of data is collected and stored, then processed to become information to be used to decision making [5].

An important characteristic of the data-driven economy is that fast decisions make a difference [3]. Then, the information is required almost in real time, which is difficult to achieve through batching processing. Another approach to process and provide data to decision making processes is Data Stream Processing (DSP), where the data is processed as soon it is generated, with a major reduction from the time between generating the data and the time it becomes usable information [5].

Advances on the Internet of Things (IoT) are creating scenarios where the data used for decision making is generated at the edges of the network [16]. Since most DSP applications explore the cloud infrastructure for deployment, data generated at the edges introduces a challenge to cloud processing, due to the fact that sending large amounts of data through the Internet faces network congestion introducing delay to the processing tasks [14]. While the advances on IoT introduce challenges, they also present themselves as a solution. IoT resources have constrained

but non-negligible computational power and can be explored to offload computing from the cloud to the edges of the network [5, 21, 22].

A solution for the deployment of DSP applications at the edges of the network would benefit healthcare applications, where users with wearable devices are continuously monitored to identify health problems and contact emergency systems as fast as possible [10]. Another application that would benefit from fast responses is natural disaster management, where sensors and cameras would make it possible to identify critical areas that require more attention and schedule rescue teams to such areas [8]. There are many other cases, such as, traffic monitoring, camera surveillance, energy grid management, etc. All these applications require fast response time [9]. Hence, there is a growing demand for solutions to schedule DSP applications exploring the low network latency provided by edge resources and high computational power from cloud to provide near real-time information.

6.1.1 Thesis Contributions

This thesis described algorithmic solutions for addressing the joint optimization of operator placement and parallelism for DSP applications deployed on infrastructure that combines both cloud and edge computing resources; where edge computing comprises IoT and Micro Datacenters (MDs) resources. We proposed solutions that seek to explore the operator parallelism for scheduling operators aiming to improve performance and Quality of Service (QoS) metrics such as application end-to-end latency, throughput, and monetary costs.

The development of the proposed solutions stemmed from investigating other approaches in the literature that aim to solve the operator placement problem for DSP applications. This investigation allowed us to identify common characteristics both for the application and operators, such as operator selectivity, data transformation pattern, probability of flow distribution, and how the operator parallelism is addressed. Moreover, the investigation also enabled us to identify how existing work in the literature considers the operator's requirements.

Beyond the elements accounted for in the literature, the investigation showed limitations, such as a restricted operator parallelism model and simplistic application and infrastructure models. Another important aspect is that despite the fact that some approaches address cloud-edge infrastructure, most of the approaches are still cloud-focused, even for DSP applications targeting IoT scenarios. This thesis proposed two solutions for the *operator placement problem*, known to be NP-Hard [28], to tackle existing limitations of the state-of-the-art.

Both of the proposed solutions rely on a Mixed Integer Linear Programming (MILP) model that aims to find optimal placements for DSP applications. The proposed MILP model has a detailed application model with operator selectivity, data transformation model, probability of flow distribution, and application requirements. Along with a detailed application model, there is an elaborate infrastructure model that accounts for the heterogeneity of cloud-edge infrastructure, CPU and memory capacity, and the resources' clock speed. The proposed model focuses on exploring edge resources, by splitting an operator into multiple replicas, each of which processing a fraction of the overall load of the operator. Thus, each replica's requirements are computed as a fraction of the operators' overall requirements as a function of the replica's load and the speed of the selected resource.

The proposed model explores edge resources by evaluating the trade-off between offloading the operator as multiple replicas to constrained devices with low network latency versus exploring cloud resources with high computational power but latency-distant from the data sources. Moreover, the model accounts for the network congestion between each layer of the cloud-edge

infrastructure (IoT, MD, and cloud), that usually crosses the Internet. Beyond computing the operator placement, the proposed model decides how the created replicas should communicate with each other, considering the communication pattern between the DSP application graph operators.

Usually, DSP applications explore some sort of public infrastructure for deployment. Therefore, monetary cost is usually a metric that needs to be taken into consideration. In this thesis we considered cloud-edge infrastructure as a combination of public and private infrastructure, where IoT resources belong to private infrastructure and MD and clouds are public. Then, we embedded a cost model with elements obtained from AWS Fargate [71], AWS Direct Connection [72], AWS IoT Core [77], and AWS Private Links [78]. We selected Amazon AWS services because Amazon is a major Infrastructure as a Service (IaaS) provider.

Both of the proposed solutions were developed on top of the proposed model. The first solution, named Cloud-Edge data Stream Placement (CESP), seeks an optimal operator placement, with a multi-objective optimization function to reduce the application end-to-end latency and deployment costs. We understand that application end-to-end latency and deployment costs have equal importance on the evaluated context, but the importance could be shifted either way. The second solution, called Cloud-Edge data Stream Placement with Resource Selection (CESP–RS), is an extension of CESP, addressing the optimal solution's scalability issues, which are a major problem when addressing a cloud-edge infrastructure with numerous geographically distributed resources.

When compared with a solution that focuses on cloud deployment only, CESP is able to reduce the application end-to-end latency by $\simeq$ 80% and the deployment costs by $\simeq$ 30%. A discussion was provided on whether the obtained benefits come from the usage of IoT resources only or the whole edge infrastructure. According to obtained results, the usage of MDs improves both application end-to-end latency and deployment costs when compared with a solution that explores a combination of IoT and cloud resources. On the one hand, the use of resources on all cloud-edge infrastructure layer provides benefits. On the other hand, this creates a very large search space and complicates a deployment. To this end, we proposed CESP–RS, which obtains deployments for DSP applications $\simeq$ 94% faster than CESP, with only $\simeq$ 12% degradation of optimal deployments. Regarding the comparison against Taneja's state-of-the-art approach, CESP–RS is able to provide $\simeq$ 91% reduction in the end-to-end application latency, and on most cases $\simeq$ 18% reduction on deployment costs. On the cases where CESP–RS did not provide cheaper deployment costs, it still provided $\simeq$ 83% reduction on the application end-to-end latency.

6.2 Future Directions

The demand for deploying DSP applications onto cloud-edge infrastructure will continue to rise over the next few years, but as technology advances, it will require performance improvements. In this section we highlight some of the areas in which this work could continue to be explored.

6.2.1 Reconfiguration Techniques

DSP applications deployed on cloud-edge resources might require adjustments to adapt to environmental changes. Two main factors require a DSP application to be reconfigured. First, edge resources are often less reliable than their cloud counterparts and generally powered by renewable and intermittent power sources [83], making them more prone to failures than cloud

or MD resources. Second, depending on the scenario in which the DSP application is deployed, load variations are common. Therefore it is required to explore either vertical or horizontal elasticity to reconfigure the application [84, 85].

There are already some efforts in the literature regarding the reconfiguration of DSP applications [67, 86, 87], but there is much to be explored such: as migrating operators and guaranteeing fault-tolerance; migration processes that do not affect other, non-migrated, operators; and techniques to evaluate the benefits of the migration. Similar to deployment solutions, reconfiguration solutions are mainly focused on cloud resources, leaving a large space for optimization on the cloud-edge infrastructure. Even further, solutions usually focus only on the application for optimization, creating more or less replicas, using more or less resources, but neglect network optimization, which on cloud-edge scenarios has a major impact on the performance of the application [29].

6.2.2 Real Infrastructure Deployment

The goal of any mathematical model is to represent the reality with the most accuracy as possible, and with that in mind, we created a model to predict the throughput of DSP applications, end evaluated it under a real environment. Results are available in [40], and our model was able to predict the throughput with $\simeq 99\%$ accuracy. The prediction model was used as cornerstone to develop the scheduling model proposed in this thesis and its implementations (CESP and CESP–RS).

Despite using a prediction model as cornerstone for the development of the scheduling model proposed in Section 3.3, some aspects needed to be adapted to obtain a schedule proposal. We developed it trying to replicate as much as possible all the application and infrastructure elements that best represent the reality. But it is undeniable that there is a gap between a simulated environment and a real infrastructure, with other applications sharing network and computational resources, different isolation techniques, and so on.

In this way, a natural continuation of this work is to evaluate the proposed model and solutions on a real infrastructure, with different deployment frameworks such as those presented in Section 2.3 (*e.g.* Apache Storm, Apache Flink) or isolation technologies such as containers, Virtual Machine (VM), etc. Such evaluation would allow identifying possible limitations of the model regarding the reality and how to fine-tune the model to overcome these limitations.

6.2.3 Stateful Operators

As mentioned early in this thesis, operators on DSP applications can be classified according to their state, where an operator can be *stateless*, in which case it does not maintain any state between executions; *partitioned stateful* where a given data structure maintains state for each down stream based on a partitioning key, and *stateful* where no particular structure is required [13]. The state of the operator is a property that was not addressed in this thesis.

The state of the operator is an important aspect that needs to be taken into consideration. It adds more complexity to the operator placement problem, especially with operator replication. Operators that maintain state or a partial state require some level of knowledge about the overall data arriving in the application, and with replication, not all the data will cross through the operator, and the challenge relies upon deciding how to split such operators, and at the same time maintain their states updated.

6.2.4 Scalability

Although we proposed CESP–RS, a solution to improve the scalability of CESP, further work is required. The proposed heuristic is able to obtain fast placements at $\simeq 12\%$ reduction on the quality of the optimal solution, but we can explore solutions to reduce even further this quality degradation. However, a resource technique based on other characteristics, not just the worst fit resource, might reduce the optimal solution's degradation quality.

Moreover, we did not evaluate the infrastructure fragmentation over a long period of time and multiple applications deployment. Then, a heuristic that is both able to reduce the search space and provide resources to reduce the resource fragmentation on the infrastructure would be beneficial both for the application user/owner and the IaaS. The application user/owner would be able to explore latency-close resources and, therefore, obtain a better application end-to-end latency. The IaaS would be able to deploy more applications simultaneously.

6.2.5 Machine Learning for DSP Placement and Reconfiguration

There is a recent trend exploring machine learning solutions to the most diverse problems, and this is reflected in solutions for operator placement and reconfiguration. Then, the natural direction to extend the current work is to identify how DSP applications could benefit from using machine learning for placement and reconfiguration. Current work offers good insights into how machine learning can be explored in this context [67, 88, 89, 90].

It is possible to create deployment models using machine learning or deciding how and where to deploy operators within the infrastructure, or to reconfigure each operator or the whole application. One can create prediction models using machine learning to guide whether the application should be reconfigured (*e.g.*, scale in/out). We performed preliminary work on reconfiguration where classic reinforcement learning techniques were used to change the initial operator placement [67]. Another interesting aspect in which machine learning solutions could be employed is to reduce the search space evaluated for optimal solutions with minimal quality degradation.

Bibliography

[1] Thi Mai Trang Nguyen et al. "SDN-based Wi-Fi Direct clustering for cloud access in campus networks." In: *Annals of Telecommunications* 73.3-4 (2018), pp. 239–249.

[2] Hind Bangui et al. "Moving to the edge-cloud-of-things: recent advances and future research directions." In: *Electronics* 7.11 (2018), p. 309.

[3] Radhya Sahal, John G Breslin, and Muhammad Intizar Ali. "Big data and stream processing platforms for Industry 4.0 requirements mapping for a predictive maintenance use case." In: *Journal of Manufacturing Systems* 54 (2020), pp. 138–151.

[4] Rajiv Ranjan et al. "Orchestrating Bigdata Analysis Workflows." In: *IEEE Cloud Computing* 4.3 (2017), pp. 20–28.

[5] Valeria Cardellini et al. "Joint operator replication and placement optimization for distributed streaming applications." In: (2017), pp. 263–270.

[6] Saeed Shahrivari. "Beyond batch processing: towards real-time and streaming big data." In: *Computers* 3.4 (2014), pp. 117–129.

[7] Matthew. "Big Data: Are you ready for blast-off?" In: *BBC News* (2014). URL: `https://www.bbc.com/news/business-26383058`.

[8] Avrilia Floratou et al. "Dhalion: self-regulating stream processing in heron." In: *Proceedings of the VLDB Endowment* 10.12 (2017), pp. 1825–1836.

[9] Paulo Ferrão, Hélder Marques, and Hervé Paulino. "Stream Processing on Hybrid CPU/Intel® Xeon Phi™ Systems." In: (2018), pp. 796–810.

[10] Fuyuan Xiao and Masayoshi Aritsugi. "An adaptive parallel processing strategy for complex event processing systems over data streams in wireless sensor networks." In: *Sensors* 18.11 (2018), p. 3732.

[11] Valeria Cardellini et al. "Optimal operator deployment and replication for elastic distributed data stream processing." In: *Concurrency and Computation: Practice and Experience* 30.9 (2018), e4334.

[12] Valeria Cardellini et al. "Optimal operator replication and placement for distributed stream processing systems." In: *ACM SIGMETRICS Performance Evaluation Review* 44.4 (2017), pp. 11–22.

[13] Marcos Dias de Assuncao, Alexandre Da Silva Veith, and Rajkumar Buyya. "Resource elasticity for distributed data stream processing: A survey and future directions." In: *CoRR, abs/1709.01363* (2017).

[14] Xinwei Fu et al. "Edgewise: a better stream processing engine for the edge." In: (2019), pp. 929–946.

[15] Valeria Cardellini et al. “Optimal operator placement for distributed stream processing applications.” In: (2016), pp. 69–80.

[16] Tanissia Djemai et al. “A Discrete Particle Swarm Optimization approach for Energy-efficient IoT services placement over Fog infrastructures.” In: (2019), pp. 32–40.

[17] Dave Evans. “The internet of things: How the next evolution of the internet is changing everything.” In: *CISCO white paper* 1.2011 (2011), pp. 1–11.

[18] Luiz Angelo Steffenel. “Improving the Performance of Fog Computing through the use of Data Locality.” In: (2018), pp. 217–224.

[19] Shigeru Imai et al. “Airplane flight safety using error-tolerant data stream processing.” In: *IEEE Aerospace and Electronic Systems Magazine* 32.4 (2017), pp. 4–17.

[20] S. Amini, I. Gerostathopoulos, and C. Prehofer. “Big data analytics architecture for real-time traffic control.” In: (2017), pp. 710–715.

[21] Shouli Zhang et al. “Latency-Aware Deployment of IoT Services in a Cloud-Edge Environment.” In: (2019), pp. 231–236.

[22] Qinglan Peng et al. “Joint Operator Scaling and Placement for Distributed Stream Processing Applications in Edge Computing.” In: (2019), pp. 461–476.

[23] Mohammad Aazam and Eui-Nam Huh. “Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT.” In: (2015), pp. 687–694.

[24] N. Mohan and J. Kangasharju. “Edge-Fog cloud: A distributed cloud for Internet of Things computations.” In: (2016), pp. 1–6.

[25] Gayashan Amarasinghe et al. “A data stream processing optimisation framework for edge computing applications.” In: (2018), pp. 91–98.

[26] Henriette Röger and Ruben Mayer. “A comprehensive survey on parallelization and elasticity in stream processing.” In: *ACM Computing Surveys (CSUR)* 52.2 (2019), pp. 1–37.

[27] Peter Pietzuch et al. “Network-aware operator placement for stream-processing systems.” In: (2006), pp. 49–49.

[28] Anne Benoit et al. “Scheduling Linear Chain Streaming Applications on Heterogeneous Systems with Failures.” In: *Future Gener. Comput. Syst.* 29.5 (July 2013), pp. 1140–1151. ISSN: 0167-739X.

[29] Wenlu Hu et al. “Quantifying the Impact of Edge Computing on Mobile Applications.” In: APSys ’16 (2016).

[30] Weisong Shi et al. “Edge computing: Vision and challenges.” In: *IEEE internet of things journal* 3.5 (2016), pp. 637–646.

[31] Tarek Elgamal et al. “Droplet: Distributed operator placement for iot applications spanning edge and cloud resources.” In: (2018), pp. 1–8.

[32] Laurent Prosperi et al. “Planner: cost-efficient execution plans placement for uniform stream analytics on edge and cloud.” In: (2018), pp. 42–51.

[33] Phu Lai et al. “Edge user allocation with dynamic quality of service.” In: (2019), pp. 86–101.

[34] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. "Edge affinity-based management of applications in fog computing environments." In: (2019), pp. 61–70.

[35] Asad Javed et al. "IoTEF: A Federated Edge-Cloud Architecture for Fault-Tolerant IoT Applications." In: *Journal of Grid Computing* (2020), pp. 1–24.

[36] Wei Yu et al. "A survey on the edge computing for the Internet of Things." In: *IEEE access* 6 (2017), pp. 6900–6919.

[37] Abhishek Tiwari et al. "Reconfigurable Streaming for the Mobile Edge." In: (2019), pp. 153–158.

[38] Zhengping Qian et al. "Timestream: Reliable stream computation in the cloud." In: (2013), pp. 1–14.

[39] Anshu Shukla, Shilpa Chaturvedi, and Yogesh Simmhan. "Riotbench: A real-time iot benchmark for distributed stream processing platforms. CoRR abs/1701.08530 (2017)." In: *arxiv. org/abs/1701.08530* (2017).

[40] F. R. d. Souza, M. D. d. Assunção, and E. Caron. "A Throughput Model for Data Stream Processing on Fog Computing." In: (2019), pp. 969–975.

[41] F. R. d. Souza et al. *An Optimal Model for Optimizing the Placement and Parallelism of Data Stream Processing Applications on Cloud-Edge Computing*. Porto, Portugal, 2020.

[42] F. R. d. Souza et al. *An Scalable Joint Optimization of Placement and Parallelism of Data Stream Processing Applications on Cloud-Edge Infrastructure*. Dubai, United Arab Emirates, 2020.

[43] Roya Golchay et al. "Spontaneous proximity clouds: Making mobile devices to collaborate for resource and data sharing." In: (2016), pp. 480–489.

[44] Bugra Gedik, Habibe G Özsema, and Özcan Öztürk. "Pipelined fission for stream programs with dynamic selectivity and partitioned state." In: *Journal of Parallel and Distributed Computing* 96 (2016), pp. 106–120.

[45] Xunyun Liu and Rajkumar Buyya. "Performance-oriented deployment of streaming applications on cloud." In: *IEEE Transactions on Big Data* 5.1 (2017), pp. 46–59.

[46] Yuzhe Tang and Bugra Gedik. "Autopipelining for data stream processing." In: *IEEE Transactions on Parallel and Distributed Systems* 24.12 (2012), pp. 2344–2354.

[47] *Apache Edgent*. Retired Project Website. 2020. URL: `https://github.com/apache/incubator-retired-edgent`.

[48] Samantha Chan. *Apache Quarks, Watson, and Streaming Analytics: Saving the World, One Smart Sprinkler at a Time*. Bluemix Blog. 2016. URL: `https://www.ibm.com/blogs/bluemix/2016/06/better-analytics-with-apache-quarks/`.

[49] *Apache MiNiFi*. Project Website. 2020. URL: `https://nifi.apache.org/minifi/index.html`.

[50] *Kafka Streams*. Project Website. 2020. URL: `https://kafka.apache.org/documentation/streams/`.

[51] *Node-RED*. Project Website. 2020. URL: `https://nodered.org/`.

[52] Valeria Cardellini et al. "Decentralized self-adaptation for elastic data stream processing." In: *Future Generation Computer Systems* 87 (2018), pp. 171–185.

[53] Farah Aït Salaht, Frédéric Desprez, and Adrien Lebre. "An Overview of Service Placement Problem in Fog and Edge Computing." In: *ACM Computing Surveys* 53.3 (June 2020). ISSN: 0360-0300. DOI: 10.1145/3391196. URL: https://doi.org/10.1145/3391196.

[54] Cheol-Ho Hong and Blesson Varghese. "Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms." In: *ACM Computing Surveys* 52.5 (Sept. 2019). ISSN: 0360-0300. DOI: 10.1145/3326066. URL: https://doi.org/10.1145/3326066.

[55] Walid Aljoby et al. "On SDN-Enabled Online and Dynamic Bandwidth Allocation for Stream Analytics." In: (2018), pp. 209–219.

[56] Yating Zhang et al. "On Cost Efficient Dataflow Computing Program Deployment in SDN Managed Distributed Computing Environment." In: (2017), pp. 27–36.

[57] Buğra Gedik et al. "Elastic scaling for data stream processing." In: *IEEE Transactions on Parallel and Distributed Systems* 25.6 (2013), pp. 1447–1463.

[58] Christoph Hochreiner et al. "Elastic stream processing for the internet of things." In: (2016), pp. 100–107.

[59] Radu Tudoran et al. "Jetstream: Enabling high throughput live event streaming on multi-site clouds." In: *Future Generation Computer Systems* 54 (2016), pp. 274–291.

[60] Raphael Eidenbenz and Thomas Locher. "Task allocation for distributed stream processing." In: (2016), pp. 1–9.

[61] Mohit Taneja and Alan Davy. "Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm." In: (2017), pp. 1222–1228.

[62] Thomas Hiessl et al. "Optimal placement of stream processing operators in the fog." In: (2019), pp. 1–10.

[63] Lin Gu et al. "A general communication cost optimization framework for big data stream processing in geo-distributed data centers." In: *IEEE Transactions on Computers* 65.1 (2015), pp. 19–29.

[64] Walid AY Aljoby, Tom ZJ Fu, and Richard TB Ma. "Impacts of task placement and bandwidth allocation on stream analytics." In: (2017), pp. 1–6.

[65] Tri Minh Truong et al. "Performance Analysis of Large-Scale Distributed Stream Processing Systems on the Cloud." In: (2018), pp. 754–761.

[66] Leila Eskandari et al. "T3-Scheduler: A topology and traffic aware two-level scheduler for stream processing systems in a heterogeneous cluster." In: *Future Generation Computer Systems* 89 (2018), pp. 617–632.

[67] Alexandre da Silva Veith et al. "Multi-Objective Reinforcement Learning for Reconfiguring Data Stream Analytics on Edge Computing." In: (2019), pp. 1–10.

[68] Xiang Ni et al. "Generalizable Resource Allocation in Stream Processing via Deep Reinforcement Learning." In: 34.01 (2020), pp. 857–864.

[69] Gabriele Russo Russo, Valeria Cardellini, and Francesco Lo Presti. "Reinforcement learning based policies for elastic stream processing on heterogeneous resources." In: (2019), pp. 31–42.

[70] X. Liu and R. Buyya. "Performance-Oriented Deployment of Streaming Applications on Cloud." In: *IEEE Tr. on Big Data* 5.1 (2019), pp. 46–59. ISSN: 2372-2096.

[71] *AWS Fargate*. https://aws.amazon.com/fargate/. Accessed: 2020-09-21.

[72] *AWS Direct Connect*. https://aws.amazon.com/directconnect/. Accessed: 2020-09-21.

[73] Frank Dabek et al. "Vivaldi: A decentralized network coordinate system." In: 34.4 (2004), pp. 15–26.

[74] Diego Kreutz et al. "Software-defined networking: A comprehensive survey." In: *Proc. of the IEEE* 103.1 (2015), pp. 14–76.

[75] Deepak Puthal et al. "Secure and Sustainable Load Balancing of Edge Data Centers in Fog Computing." In: *IEEE Communications Magazine* 56.5 (2018), pp. 60–65.

[76] Hamidreza Arkian et al. "An Experiment-Driven Performance Model of Stream Processing Operators in Fog Computing Environments." In: (2020).

[77] *AWS IoT Core*. https://aws.amazon.com/iot-core/. Accessed: 2020-09-21.

[78] *AWS PrivateLinke*. https://aws.amazon.com/privatelink/. Accessed: 2020-09-21.

[79] J. Gedeon et al. "From Cell Towers to Smart Street Lamps: Placing Cloudlets on Existing Urban Infrastructures." In: (2018), pp. 187–202.

[80] Steffen Zeuch et al. "Analyzing Efficient Stream Processing on Modern Hardware." In: *Proc. VLDB Endow.* 12.5 (Jan. 2019), pp. 516–530. ISSN: 2150-8097. DOI: 10.14778/3303753.3303758. URL: https://doi.org/10.14778/3303753.3303758.

[81] A. Finamore et al. "Experiences of Internet traffic monitoring with tstat." In: *IEEE Network* 25.3 (2011), pp. 8–14.

[82] A. Callado et al. "A Survey on Internet Traffic Identification." In: *IEEE Communications Surveys Tutorials* 11.3 (2009), pp. 37–52.

[83] Minxian Xu et al. *Green-aware Mobile Edge Computing for IoT: Challenges, Solutions and Future Directions*. 2020. arXiv: 2009.03598 [cs.DC].

[84] Dawei Sun et al. "Rethinking elastic online scheduling of big data streaming applications over high-velocity continuous data streams." In: *The Journal of Supercomputing* 74.2 (2018), pp. 615–636.

[85] L. Xu, B. Peng, and I. Gupta. "Stela: Enabling Stream Processing Systems to Scale-in and Scale-out On-demand." In: (2016), pp. 22–31.

[86] Tiziano De Matteis and Gabriele Mencagli. "Proactive elasticity and energy awareness in data stream processing." In: *Journal of Systems and Software* 127 (2017), pp. 302–319.

[87] Thomas Heinze et al. "Online parameter optimization for elastic data stream processing." In: (2015), pp. 276–287.

[88] Valeria Cardellini et al. "Auto-scaling in data stream processing applications: A model-based reinforcement learning approach." In: (2017), pp. 97–110.

[89] M. Endler et al. "Towards stream-based reasoning and machine learning for IoT applications." In: (2017), pp. 202–209.

[90] G. Pal, G. Li, and K. Atkinson. "Big Data Real Time Ingestion and Machine Learning." In: (2018), pp. 25–31.

List of publications

Journal Publication

- **de Souza, F. R.**; Miers, C. C., Fiorese, A.; de Assunção, M. D.; Koslovski, G. P. (2019). Qvia-SDN: Towards QOS-Aware Virtual Infrastructure Allocation on SDN-based Clouds. Journal of Grid Computing, 17(3), 447-472. 2019. (Core Ranking B)

Publications in International Conferences

- **de Souza, F. R.**; de Assunção, M. D.; Caron, E. A Throughput Model for Data Stream Processing on Fog Computing. International Conference on High Performance Computing & Simulation (HPCS 2019), Dublin, Ireland, July 2019. (Core Ranking B)
- da Silva Veith, A.; **de Souza, F. R.**; Santos dos Anjos, J. C.; de Assunção M. D.; Lefèvre, L. Multi-Objective Reinforcement Learning for Reconfiguration of Data Analytics on Edge Computing. International Conference on Parallel Processing (ICPP 2019), Kyoto, Japan, August 2019. (Core Ranking A)
- **de Souza, F. R.**; da Silva Veith, A.; de Assunção, M. D.; Caron, E. An Optimal Model for Optimizing the Placement and Parallelism of Data Stream Processing Applications on Cloud-Edge Computing. IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2020), Porto, Portugal, September 2020. (Core Ranking B)
- **de Souza, F. R.**; da Silva Veith, A.; de Assunção, M. D.; Caron, E. Scalable Joint Optimization of Placement and Parallelism of Data Stream Processing Applications on Cloud-Edge Infrastructure. 18th International Conference on Service Oriented Computing (ICSOC 2020), Dubai, United Arab Emirates, December 2020. (Core Ranking A)

Publication in a National Conference

- Raugust, A. S.; **de Souza, F. R.**; Pillon, M. A.; Miers, C. C.; Koslovski, G. P. Alocação de Infraestruturas Virtuais Confiáveis em Múltiplos Provedores IaaS. Simpósio Brasileiro de Redes de Computadores (SBRC), 36, 2018.