Towards a Security-Aware Deployment of Data Streaming Applications in Fog Computing

Gabriele Russo Russo, Valeria Cardellini, Francesco Lo Presti, and Matteo Nardelli

1 Introduction

In the recent few years we have witnessed a worldwide explosion of the volume of daily produced data, fostered by the spread of sensors, wearable devices, and smartphones capable of collecting data about their surrounding environment and our everyday life. Today, all this information plays a key role in our society, and has become a strategical asset for institutions, companies, and scientists. The analysis of collected data allows to extract information useful for supporting decision-making, e.g., gathering new insights by identifying patterns or making predictions based on past observations.

In this context, the ability of efficiently collecting, storing, and analyzing data has become a strategic advantage. Nevertheless, the aforementioned growth has made data processing challenging from a computational point of view, and led to the development of efficient algorithms, tools, and frameworks for dealing with data (e.g., the *Map-Reduce* paradigm, and associated frameworks like *Apache Hadoop*).

Special interest has been devoted to *real-time* data analytics, which requires systems able to process data as soon as they are collected. This is critical in many application domains, e.g., in network attack detection, where monitoring information about the incoming traffic should be processed with very low latency. In this context, a primary role is played by distributed *Data Stream Processing* (DSP) systems, which allow to process unbounded sequences of data (i.e., streams),

G. Russo Russo · V. Cardellini (✉) · F. Lo Presti · M. Nardelli
Department of Civil Engineering and Computer Science Engineering, University of Rome Tor Vergata, Rome, Italy
e-mail: russo.russo@ing.uniroma2.it; cardellini@ing.uniroma2.it; lopresti@info.uniroma2.it; nardelli@ing.uniroma2.it

W. Chang, J. Wu (eds.), *Fog/Edge Computing For Security, Privacy, and Applications*, Advances in Information Security 83,
https://doi.org/10.1007/978-3-030-57328-7_14

flowing at very high rates, exploiting a multitude of computing nodes to spread the computation.

In the effort to further reduce processing latency with respect to data producers and consumers, which are often located at the edge of the network, recently DSP systems have been shifted from traditional cloud data centers to fog computing environments. By deploying applications in such geographically distributed infrastructures, latency reduction is achieved at the cost of handling increased heterogeneity, constrained computational and network resources, and a larger number of security concerns. These challenges especially impact the application *placement* problem, that is the problem of determining the set of computing nodes where application components are deployed and executed. This choice is indeed critical for achieving the expected Quality-of-Service (QoS), while minimizing application operating costs.

The placement problem for DSP applications has been widely investigated in literature, in the context of both traditional cloud scenarios, and geographically distributed environments (e.g., fog computing), exploiting a variety of methodologies. Existing solutions take into account performance-oriented characterizations of both the application and the computing infrastructure, in order to determine a placement scheme that optimizes one or more QoS metrics (e.g., system response time, or deployment monetary cost). Unfortunately, most the existing approaches neglect the security- and privacy-related concerns that inevitably arise when DSP applications are deployed in fog-like environments, where they may rely on a mixture of wired and wireless network links, and computing resources characterized by different software/hardware configurations, possibly acquired from multiple providers.

In this chapter, to overcome the limitations of existing placement optimization solutions, we present a simple yet quite general approach to account security related aspects. To this end, we introduce a formalism to specify application requirements and describe infrastructure features and capabilities. We also define associated metrics that capture how well different placement solutions match the specified application requirements, and allow us then to seamlessly integrate security related requirements in the overall optimization scheme. Our contributions are as follows.

- We present a formalism for specifying security-related application requirements. The idea is to represent application requirements as a forest of AND-OR requirements trees, each capturing a specific security requirement, e.g., privacy, isolation. At the same time we show how this formalism can be used to derive several requirement satisfaction metrics.
- We introduce the notion of operator and data stream configurations which define the set of security related configurations which satisfy the application requirements. This is paralleled by the notion of configurations that the infrastructure computing nodes and data links can support. The concept of configurations is the basis around which stakeholders can reason about application requirements and infrastructure characteristics and lay out the foundation of our deployment problem formulation.

- Finally, we integrate the aforementioned requirements, configurations and metrics in the placement optimization problem, which is formulated as Integer Linear Programming (ILP), and also accounts for other application non-functional requirements, e.g., response time and cost, and present the *Security-Aware DSP Placement* (SDP) problem. Focusing on a realistic case study, we show how SDP allows us to compute trade-offs between performance-based metrics, deployment cost, and security-related requirements satisfaction.

1.1 Organization of the Chapter

In the next section, we will provide an overview of the basic concepts of DSP, and the challenges faced when deploying this kind of applications in fog environments, especially as regards security concerns. In Sect. 3, we present a formalism for specifying the application security-related requirements for deployment. In Sect. 4, we explain how we model the application placement problem, including the application and infrastructure model, and the associated QoS and cost metrics. The resulting problem formulation is presented in Sect. 5, along with an illustrative example of how it can be applied. We discuss the benefits and limitations of the presented approach in Sect. 6, and conclude in Sect. 7.

2 Background

In this section, we provide an overview of the main concepts, challenges, and research directions related to DSP, and the deployment of DSP applications in the fog environment. First, we describe in Sect. 2.1 the basic concepts and main challenges related to the deployment of DSP applications. Then, in Sect. 2.2, we focus on research works that address the placement of DSP applications. Finally, in Sect. 2.3, we describe works that deal with security and privacy issues in the DSP domain.

2.1 Data Stream Processing: Basic Concepts and Challenges

A DSP application consists of a network of processing elements, called *operators*, connected by data streams. A DSP application can be represented as a directed acyclic graph (DAG), with data sources, operators, and final consumers as vertices, and streams as edges. A stream is an unbounded sequence of data items (e.g., event, tuple). Each operator is a self-contained processing element, that continuously receives incoming streams, applies a transformation on them, ranging from a simple operation (e.g., filtering, aggregation) to something more complex (e.g., applying a machine learning algorithm to detect some patterns), and generates new outgoing streams. Each data source (e.g., an IoT sensor or a message queue) generates one or more streams that feed the DSP application; differently from operators, data sources have no incoming streams. A final consumer (or sink) is a final receiver of the application streams; it can push data on a message queue, forward information to a persistent storage, or trigger the execution of some external services. Differently from operators, sinks have no outgoing streams.

An example of a DSP application related to the smart health domain, e.g., [2], is shown in Fig. 1. Data are collected by sensors on users' devices (e.g., smartphones, wrist-worn wearable devices), and sent for analysis to a DSP application. The application may carry out several kinds of processing on the data at the same time. In the example we consider, the application is used both for (i) detecting anomalies in the vital parameters monitored (e.g., skin temperature, heartbeat, and oxygen saturation in the blood) and send notifications to medical staff, and (ii) creating aggregated statistics. Users' devices push data into a message queue system, which in turn sends data as streams to the DSP system for processing. Before entering the DSP system, data can also be integrated (e.g., to merge the incoming streams into a single flow) and pre-processed (e.g., to detect duplicates). In the application DAG we can identify one source of the DSP application, four operators, and two sinks, the latter corresponding to the data consumers (medical staff and storage system). The DSP operators perform different tasks that range from aggregating data using summary statistics (the upper path in the DAG) to detecting any anomaly in the data streams (the lower path in the DAG). Although simple, this DSP application is an example of edge-native applications [56], which can take advantage of one or more of the benefits that arise from the fog/edge deployment: bandwidth scalability, low latency, enhanced privacy, and improved resiliency to WAN network failures.

DSP applications are typically deployed on either locally distributed clusters or centralized cloud data centers, which are often distant from data sources. However, pushing fast-rate data streams from sources to distant computing resources can exacerbate the load on the Internet infrastructure and introduce excessive delays experienced by DSP application users. Moreover, considering that both data sources and consumers are usually located at the network edges, a solution that allows to improve scalability and reduce network delays lies in deploying DSP applications

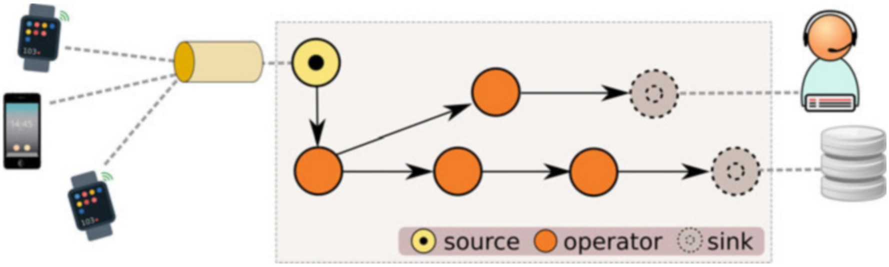


Fig. 1 Example of a *smart health* application

not only on cloud data centers but also on edge/fog computing resources. Furthermore, the deployment on edge/fog resources can also enable users to selectively control the disclosure of sensitive information (e.g., vital parameters monitored by wrist-worn devices as in the example shown in Fig. 1).

In such a distributed scenario, a relevant problem consists in determining the computing nodes that should host and execute each operator of a DSP application, aiming to optimize some QoS attributes. This problem is known in literature as the *operator placement problem* (or scheduling problem).

Figure 2 illustrates a possible placement of the DSP application shown in Fig. 1 on the computing infrastructure. Multiple operators can also be co-located on the same computing node (e.g., op_2 and op_4).

Besides the initial placement of the DSP operators, the deployment of DSP applications can also be changed at run-time, that is during the application execution, so to self-adapt it with respect to workload changes and dynamism of the edge/fog computing environment (e.g., resource constraints, network constraints in term of latency and bandwidth, resources that join or leave the system). To this end, different approaches can be applied, ranging from the exploitation of performance-enhancing techniques (e.g., operator replication by means of elastic scale-out and scale-in operations, other types of dynamic transformation of the DAG) to the run-time adaptation of the application placement. The latter can be achieved at different grains, by placing either all the DSP operators from scratch (in this case, the placement problem is solved at regular intervals, so to update the operator location) or only a subset of operators by relying on operator migration between computing resources. Determining the operator replication degree is often addressed in literature as an independent and orthogonal decision with respect to the operator placement, but in [12] we present a problem formulation that jointly optimizes the replication and placement of DSP applications. In this chapter, we assume that the operator replication degree has been set at application design

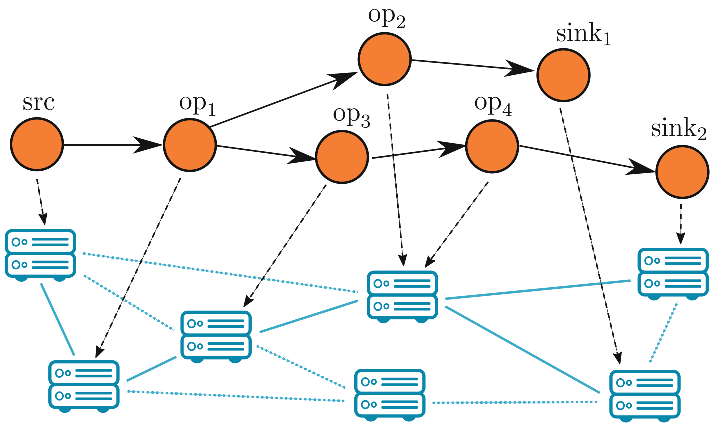


Fig. 2 Illustration of the placement problem for a DSP application

time; so, we address the initial operator placement that is, how to place to DSP operators on the computing resources at the application start. To this end, in Sect. 2.2 we focus our literature analysis on those works that address the placement of DSP applications in the fog/edge scenario, or more generally in a geo-distributed computing environment. As regards research works dealing with the run-time self-adaptive control of DSP applications, we refer the interested reader to some surveys [20, 51] that classify and review them.

2.2 Placement of DSP Applications in the Fog

The DSP placement problem has been widely investigated in literature under different modeling assumptions and optimization goals (e.g., [21, 62, 63]). We review the related works organizing them along three main dimensions, that capture one or more related facets of the problem: (1) placement goals; (2) methodologies used to define the application placement; and (3) characteristics of the distributed computing infrastructure managed by the placement solution. For a deeper analysis of the state of the art, we refer the interested reader to extensive surveys, that analyze in details the research works addressing the placement problem not only in the context of DSP applications but also for other kinds of applications deployed in the fog/edge environment [7, 8, 11, 62].

Placement Goals Existing works consider two main classes of problems: constraint satisfaction and objective function optimization. In a *constraint satisfaction* problem, we are interested in identifying a deployment solution among all the feasible ones that satisfy some given requirements (e.g., application performance). For example, Thoma et al. [63] propose an approach to restrict the set of feasible deployment by improving the expressiveness of constraints. In most cases, not all feasible deployment result in desirable application performance; therefore, most of the existing solutions optimize (i.e., minimize or maximize) a *single-objective* function or a *multiple objective* function. A single-objective optimization considers a specific and well-defined QoS metric (e.g., response time, throughput, network usage, cost). A multi-objective optimization (or Pareto optimization) aims to combine different, possibly conflicting QoS attributes and to find the set of optimal solutions (i.e., those lying on the so-called Pareto frontier). The existing solutions aim at optimizing a diversity of objectives, such as to minimize the application response time (e.g., [6, 19, 34, 52]), the inter-node traffic (e.g., [4, 22, 27, 66, 67]), the network usage (e.g., [48, 50]), or a generic cost function that can comprise different QoS metrics (e.g., [5, 13, 21, 38, 53, 65]).

Methodologies The most popular methodologies used to address the operator placement problem include mathematical programming (e.g. [5, 13, 21]), graph-theoretic approaches (e.g., [23, 35]), greedy approaches (e.g., [4, 26, 33, 36, 52, 66]), meta-heuristics (e.g., genetic algorithms [60], local search [17, 61], tabu search and simulated annealing [61], steepest descent method and tabu search [31]), as well as custom heuristics (e.g., [19, 22, 40, 46, 48, 53]).

The operator placement problem, formulated as optimization problem that takes into account the heterogeneity of application requirements and infrastructural resources, turns out to be an NP-hard problem [13]. Therefore, many research efforts focus on applying different methodologies that can solve efficiently the DSP placement problem within a feasible amount of time, even for large problem instances.

Computing Infrastructure Most of the existing solutions have been designed for a clustered environment, where network latencies are almost zero (e.g. [26, 34, 52]). Although interesting, these approaches might not be suitable for geo-distributed environments. Several works indirectly consider the network contribution by minimizing the amount of data exchanged between computing nodes (e.g., [4, 21–23, 32, 66]). For example, Eidenbenz et al. [21] propose a heuristic that minimizes processing and transfer cost, but it works only on resources with uniform capacity. Relying on a greedy best-fit heuristic, Aniello et al. [4] and Xu et al. [66] propose algorithms that minimize the inter-node traffic. Other works explicitly take into account network latencies, thus representing more suitable solutions to operate in a geo-distributed DSP system (e.g., [6, 13, 19, 29, 40, 48, 50]). Pietzuch et al. [48] and Rizou et al. [50] minimize the network usage, that is the amount of data that traverses the network at a given instant.

So far, only a limited number of works are specifically designed for placing DSP applications in fog/edge computing environments. SpanEdge [53] allows to specify which operators should be placed as close as possible to the data sources, while Arkian et al. [5] propose an integer non-linear formulation; to reduce resolution time, they linearize the problem; nevertheless, also linear formulations may suffer from scalability issues [40]. The work in [38] presents a Pareto-efficient algorithm to tackle the operator placement problem considering both the latency and energy consumption. Khare et al. [33] present an approach that first transforms any arbitrary DAG into an approximate set of linear chains, then uses a data-driven latency prediction model for co-located linear chains to drive a greedy heuristic, which determines the operator placement with the goal to minimize the maximum latency of all paths in the DAG. Peng at al. [47] jointly target the problems of DSP operator placement and replication in an edge scenario by proposing a two-stage approach that first employs a genetic algorithm for finding a solution and then uses a bottleneck-analysis based on the system queuing model to refine it.

The combination of cloud and edge resources have been also explored. For example, Ghosh et al. [28] propose a genetic algorithm meta-heuristic and show that their approach allows to achieve lower latency and more frequent feasible solutions than placing only on Cloud resources. Da Silva Vieth et al. [59] propose strategies that first decompose the application DAG, which is a series-parallel one, and then place its operators in a latency-aware manner. However, all these proposals focus on reducing the application latency, without taking into account any concern related to privacy and security.

A few edge-based stream processing systems support processing on fog/edge resources with the goal of reducing the need for costly data transfers. These systems include Cloud services such as AWS IoT Greengrass, Google Cloud IoT Edge, Microsoft Azure IoT, and research prototypes (e.g., Frontier [44]); however, they do not appear to place the DSP application over fog/edge resources by taking into account their peculiarities. On the other hand, this goal is pursed by some research efforts that extend existing open-source data stream processing systems such as Apache Storm (e.g., [4, 40, 53, 66]), mainly to show the feasibility of their approaches.

2.3 Security and Privacy in DSP

In many DSP applications, data streams carry privacy-sensitive information about users, whose confidentiality must be obviously protected throughout processing. Other applications, while not dealing with privacy-sensitive data, may carry out safety-critical tasks based on sensor-provided information (e.g., anomaly detection in a manufacturing system), where the integrity of the involved data streams must be guaranteed to avoid unintended (and possibly dangerous) application behaviors.

Guaranteeing confidentiality and integrity of data streams has become a serious challenge, especially since the availability of computing resources at the edge of the network fostered the interest for deploying DSP applications in geographically distributed infrastructures, in the aim of reducing latency. The enforcement of security and privacy policies is difficult in these environments, where the intrinsic heterogeneity, and thus the involvement of different standards and communication stacks, does not allow the application of traditional security countermeasures. This problem is particularly evident nowadays as data analytics often meet IoT scenarios, where streams originate from a multitude of potentially untrusted, distributed devices, and the need for security policy enforcement becomes critical [58].

Security and privacy issues have received limited attention in the field of distributed DSP systems so far, with research efforts being mainly devoted to application-level issues, performance, and fault-tolerance. Nonetheless, some effort has been spent investigating how to integrate *privacy-preservation* and *access control* [55] techniques in DSP systems, in order to guarantee that only authorized access to privacy-sensitive data is allowed.

Linder and Meier [37] extend the Borealis [1] streaming engine with OxR-BAC (Owner-extended Role Based Access Control), which aims at protecting the system against improper release of information, improper modification of information, and denial of service attacks. Ng et al. [43] propose a framework for privacy-preservation in data stream processing, built around the two principles of *limited disclosure* and *limited collection* of information. They design a hierarchy-based policy model and a framework to enforce privacy protection policies, and hence limit access and operation on data streams. Carminati et al. [10, 15, 16] apply Role Based Access Control [54] to DSP, relying on *secure operators* in order to replace application operators with security-aware versions. After presenting their ACStream framework in [10], in [15] they propose a query-rewriting middleware that does not target a specific underlying DSP framework.

Nehme et al. [41, 42] focus on the *continuous access control enforcement* for data streams, observing that, given the long-running nature of DSP systems, the content of the streamed data and its privacy-sensitivity may change, hence access control policies may need to be adapted dynamically as a consequence. In particular, in [42] they introduce the concept of a *security punctuation* for enforcing access control, that is a special tuple inserted directly into the data stream, allowing the data provider to attach security "metadata" to the stream. In [41], they describe FENCE, a framework for enhancing DSP systems with continuous access control enforcement through security punctuations, with limited runtime overhead.

Anh and Datta [3] focus on the problem of preserving privacy of data while stream processing is outsourced to the cloud. They present StreamForce, a framework for enforcing access control policies in presence of an untrusted cloud provider. Thoma et al. [64] propose PolyStream, a framework that allows users to cryptographically enforce access controls over streaming data on top of an unmodified DSP system. PolyStream relies on a novel use of security punctuations that enables flexible, online policy management and key distribution, with significant overhead reduction. Schilling et al. [57] focus on large-scale distributed Complex Event Processing systems, proposing access control consolidation mechanisms in order to ensure the privacy of information even over multiple processing steps in a multi-domain, large-scale application.

A different point of view on privacy-preservation is offered by Le Quoc et al. [49]. They aim at preserving users privacy, while still supporting both information high-utility and low-latency processing. Specifically, they achieve this goal by blending together two different approaches, namely, sampling (used for approximate computation) and randomized response (used for privacy-preserving analytics).

Recently, Burkhalter et al. [9] focused on the special class of applications dealing with time series data. They propose TimeCrypt, which provides scalable, real-time analytics over large volumes of encrypted time series data, by allowing users to define expressive data access and privacy policies, and enforcing them cryptographically.

The number of works that deal with system- or network-level security aspects in the context of DSP is significantly smaller. Fisher and Hancke [24] consider the network-level challenges of transmitting privacy-sensitive data streams from sensors to the processing servers. In particular, they investigate the use of the Datagram Transport Layer Security protocol, compared with the more popular Transport Layer Security protocol. Havet et al. [30] propose *SecureStreams*, a reactive framework that combines combines a high-level dataflow programming model with low-level Intel *software guard extensions* (SGX) in order to guarantee privacy and integrity of the processed data. Park et al. [45] focus on the scenario of running stream analytics on untrusted, resource-constrained devices at the edge of the network. They present StreamBox-TZ, a stream analytics engine that offers strong data security and verifiable results, by isolating computation in a Trusted Execution Environment (TEE). In particular, StreamBox-TZ relies on a data plane designed and optimized for a TEE based on ARM TrustZone.

A different approach is proposed by Chaturvedi and Simmhan [18], who apply *Moving Target Defense* (MTD) [68] techniques to protect a DSP platform; the key idea is to introduce system configuration variability at run-time so that any prior information available to an attacker becomes hardily usable. In particular, they implement several MTD mechanisms (e.g., migrating operators periodically over available computing nodes, altering the used port numbers, modifying the application graph by means of "dummy" operators), and show the feasibility of the approach by integrating them in Apache Storm.

At a higher level of abstraction, independently of the specific DSP framework in use and the possibly associated privacy-preservation mechanisms, security and privacy concerns also impact the choices made for initially deploying DSP applications over distributed infrastructures, i.e., the placement problem. As explained above, this problem has been extensively studied, but so far only performance and cost aspects have been considered in the context of DSP. Security-aware deployment and scheduling strategies have been proposed instead targeting other kinds of fog applications (e.g., [25] and [58]). In this chapter, we aim to fill the existing gap regarding the DSP placement problem, and the consideration of privacy and security for stream analytics applications in the fog. Although we specifically focus on the *initial* placement problem, the approach we will present can be applied for updating the application deployment at run-time as well, e.g., following a MTD strategy as suggested in [18].

3 Modeling Security-Related Requirements

Traditional strategies for deploying DSP application over distributed infrastructures aim at optimizing one or more performance metrics, e.g., application response time, or throughput. Some of them also account for the monetary cost of the computing resources chosen for running the application, assuming, e.g., a typical pay-as-you-go cost model. The recent trend of shifting the data processing applications towards the edge of the network, closer to the data producers, often forces DSP applications to be deployed in a less "trusted" environment, compared, e.g., to cloud data centers. In this new scenario, application deployment strategies should therefore account for security-related aspects in addition to the other well-known functional and non-functional metrics and stakeholders should be able to specify a set of requirements and/or objectives that involve these additional non-functional aspects.

Unfortunately, although several solutions have been proposed in literature, how stakeholders should express these requirements in a *standardized* way remains an open question. In the remainder of this section, we describe a simple yet powerful technique to formalize and organize the requirements of a DSP application with respect to the underlying computing and network infrastructure by which it is hosted.

3.1 Requirement Categories and Objectives

For a DSP application we might need to specify its security, privacy, or reliability requirements, in addition to commonly adopted performance and cost objectives. The application requirements could include a broad range of different aspects, including software or network configuration, hardware capabilities, location of the computing resources. Formally, we assume that application requirements are organized into different *Requirement Categories* (RCs), denoting with Ω_{RC} the set of all the considered categories. An application may hence exhibit requirements related to one or more RCs. Clearly, the relative relevance of each RC depends on the specific application. Whilst some RCs may contain critical requirements for application operation, other RCs may simply represent application "preferences" with respect to its running environment. In the following, we will show how the different importance of each requirement will impact the optimization problem we present.

With each Requirement Category ω, we associate a set of *Requirement Objectives* (ROs), denoted as Ω^{ω}_{RO}. ROs represent specific properties of the computing infrastructure or the network (e.g., "type of operating system", "available encryption libraries", "wired/wireless network connectivity"). While RCs are mere abstractions, representing a collection of similar or related requirements, ROs represent concrete properties which can be evaluated in order to assess whether, e.g., a certain computing node satisfies the application needs. Specifically, given a RO ρ, we denote the set of values ρ can be associated with as V_{ρ}. For example, the "type of operating system" RO might be associated with the values "Linux", "Android", "Windows", "Other". Throughout this chapter, without loss of generality, we will assume that V_{ρ} is a finite set. We also find useful to define the set of the ROs comprised by all the RCs, that is $\Omega_{RO} = \bigcup_{\omega} \Omega^{\omega}_{RO}, \forall \omega \in \Omega_{RC}$.

3.1.1 Example

Figure 3 depicts a simple hierarchy of Requirement Categories and Objectives as a tree. We organize requirements into three categories: *Runtime Environment*, *Physical Security*, and *Network*. Each RC is associated with one or more ROs. The leaf nodes of the tree reported in the figure contain all the possible values for each RO.

3.2 Requirements Forest

In our approach, application requirements are expressed by means of AND-OR trees, which are widely adopted to represent security policies or requirements (e.g., in [25, 39]). AND-OR trees allow to reduce the overall requirements to

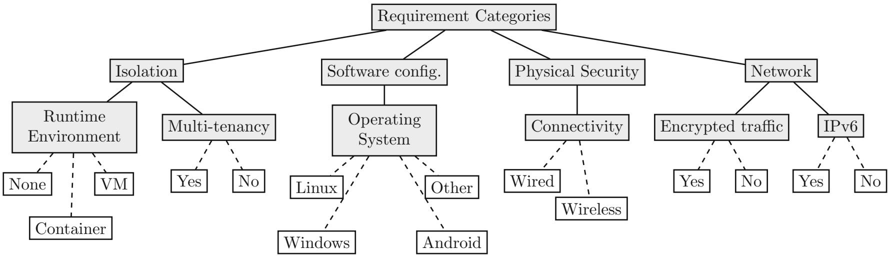


Fig. 3 Example of a hierarchy of requirement categories and requirement objectives

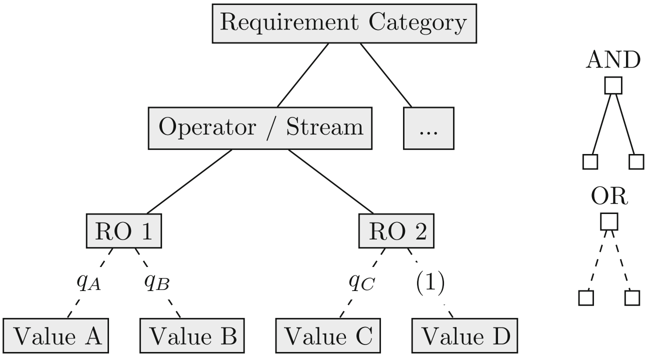


Fig. 4 Example of the AND-OR tree used to represent application requirements with respect to a requirement category. Requirements associated with different operators (or streams) and requirements involving different requirement objectives for the same application component are in an AND relationship. Whenever the application specifies multiple accepted options for the same RO, possibly providing a preference value for each option, the corresponding nodes are in an OR relationship

the conjunctions and disjunctions of “sub-requirements” (e.g., requirements coming from specific application components).

In particular, we expect an AND-OR tree $\mathcal{T}_\omega$ to be specified by the application for each category of interest $\omega \in \Omega_{RC}$. Henceforth, the overall application requirements can be formally expressed as a *forest* $\mathcal{F} = \{\mathcal{T}_\omega : \forall\, \omega \in \Omega_{RC}\}$. The structure of a generic tree $\mathcal{T}_\omega$ is illustrated in Fig. 4. As shown in the figure, the root node of each tree corresponds to the RC ω itself. The next level of the tree contains nodes associated with application components (i.e., operators or streams). Each of these nodes is the root of a subtree that represents the requirements of that specific operator (or stream). Clearly, in order to satisfy the application requirements, the requirements of every component must be satisfied. Therefore, each node at this level of the tree is in conjunction with the others.

Looking at the next level of the tree, the i-th operator (or stream) may be associated with one or more child nodes representing ROs for which a requirement is specified. We denote the set of ROs that characterize the deployment requirements of operator i as $\Omega_{RO}^{\omega,i}$. Recalling that each RO ρ takes values in a finite set V_ρ, specifying a requirement for ρ means identifying a subset of values $\tilde{V} \subseteq V_\rho$. Indeed, in our tree-based representation, RO nodes have one or more child nodes, each corresponding to a value $v \in V_\rho$ that allows to satisfy the requirement. These nodes are leaf nodes of the tree.

We further enrich our formalism by allowing leaf nodes to be associated with an optional *preference* value $q_v^i \in (0, 1]$, which is a measure of *how much* $v \in V_\rho$ satisfies the requirement of i (i.e., $q_v^i = 1$ means that v completely satisfies the requirement, whilst $0 < q_v^i < 1$ means that v is a feasible choice for the application, but with a smaller degree of satisfaction). The preference specification is optional. Wherever the preference value is not provided, complete satisfaction of the requirement is assumed (i.e., $q_v^i = 1$). Analogously, for the values $v \in V_\rho$ not appearing in the tree, which thus do not satisfy the application requirements, we will assume the preference value to be zero.

3.2.1 Example

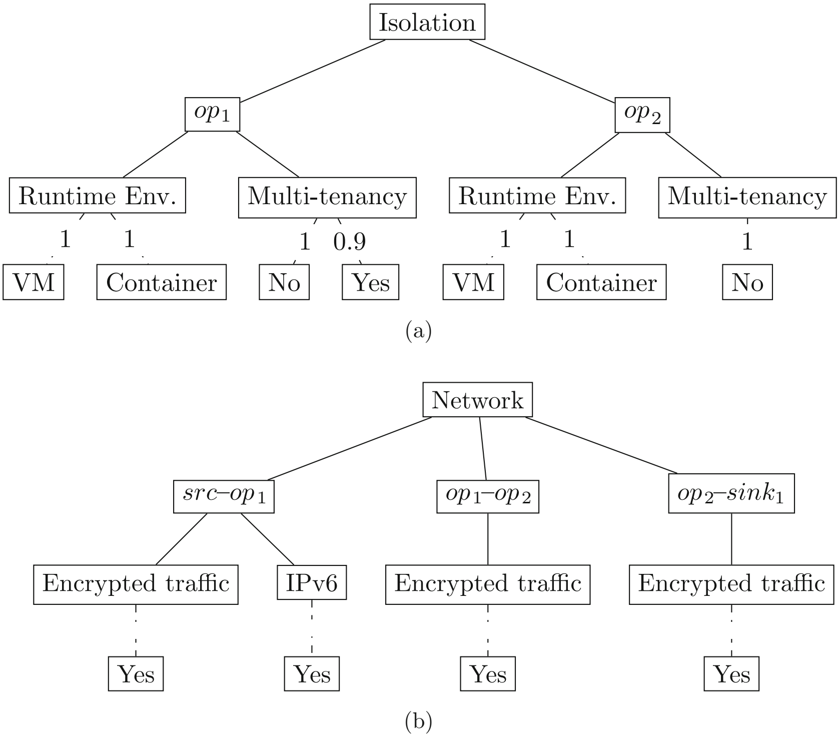
With respect to the reference DSP application depicted in Fig. 1, we show in Fig. 5 an example of the requirements that might be specified for the application. As explained, within the considered application DAG, some components are critical, because they (i) deal with privacy-sensitive information about users, and (ii) are responsible for detecting and reporting potential users' health diseases. Therefore, we expect application requirements to focus on these critical operators and streams.

For example, op_1 and op_2, which analyze users' data looking for anomalies, may require software isolation, by running either in a software container or a virtual machine. Moreover, the application may require to have those operators deployed in a dedicated node. Specifically, according to the example of Fig. 5a, op_1 requires a dedicated node, but also allows for deployment in a multi-tenant node, with a preference value smaller than 1; op_2 instead strictly requires to run in a dedicated node.

Requirements can also be specified for data streams (i.e., edges of the application DAG), imposing restrictions on the network links across which the streams can flow. Looking at Fig. 5b, we see that in the example the stream from the source to the first operator requires a network link that supports IPv6. Furthermore, all the streams in the path from the source to the first sink require data to be encrypted when traversing the network.

4 DSP Application Placement Modeling

Determining the *placement* of a DSP application means identifying, within the available computing infrastructure, the nodes where operators must be deployed and



(a)

(b)

Fig. 5 Example of requirements for the reference DSP applications

executed. This choice has clearly a major impact on application performance, cost, reliability, and security, as the available nodes may (i) be equipped with different amounts of computing resources, (ii) provide different hardware or software capabilities, and (iii) be connected to each other through different network links.

Our aim is to present a linear programming formulation for determining the optimal placement of a DSP application, which takes into account performance, cost, and security metrics. To this end, in this section we will present our model of both the application and the computing infrastructure, and we will introduce the QoS metrics we want to optimize.

4.1 System Model

In this section, we describe the system model we consider. In particular, in Sect. 4.1.1, we present how DSP applications are modeled within our optimization framework. In Sect. 4.1.2, a model of the computing infrastructure, which hosts the applications, is introduced.

4.1.1 Application Model

We represent a DSP application as a labeled directed acyclic graph (DAG) $G_{dsp} = (V_{dsp}, E_{dsp})$. The nodes in V_{dsp} represent the application operators as well as the data sources and sinks (i.e., nodes with no incoming and no outgoing link, respectively). The edges in E_{dsp} represent the data streams that flow between nodes.

We associate each node and edge in the application graph with various non-functional attributes. Specifically, for each operator i, we specify: μ_i, the average number of data units the operator can process per unit of time, on the reference processor;[1] σ_i, the *selectivity* of the operator (i.e., the average number of data units emitted by the operator per input data unit); Res_i, the amount of resources required for its execution. For the sake of simplicity, we assume Res_i to be a scalar value, representing, e.g., the number of CPU cores used by the operator. Our formulation can be easily generalized to cope with a vector of required resources, including, e.g., the amount of memory needed for execution. Similarly, we characterize the stream exchanged from operator i to j, $(i, j) \in E_{dsp}$, with its average data rate $\lambda_{(i,j)}$, and the average size (in bytes) of the data units belonging to the stream, $b_{(i,j)}$.

Moreover, we associate the application with its requirements forest $\mathcal{F}$, defined in the previous section. Each operator $\mathcal{F}$ induces a set of *feasible configurations*, i.e., the set of configurations which satisfy the operator security requirements. Formally, we define a configuration ϕ as a vector $\phi = \left(v_{\rho_1}, v_{\rho_2}, \dots, v_{\rho_N}\right)$, where $v_{\rho_i} \in V_{\rho_i}$ is a value associated with the RO ρ_i, and $\{\rho_1, \rho_2, \dots, \rho_N\} \subseteq \Omega_{RO}$.

For each operator i, $i \in V_{dsp}$, the set of feasible configuration is defined as:

$$\Phi_i = \left\{ \left(v_{\rho_1}, v_{\rho_2}, \dots\right) : \ v_{\rho_k} \in V_{\rho_k}, \forall \rho_k \in \bigcup_{\omega} \Omega_{RO}^{\omega,i} : q^i_{v_{\rho_k}} > 0 \right\} \tag{1}$$

where $\Omega_{RO}^{\omega,i}$ is the set of ROs in the RC ω that characterizes the deployment of i, and $q^i_{v_{\rho_k}} > 0$ identifies feasible values for ρ with respect to the operator requirements. An analogous definition can be given for $\Phi_{(i,j)}$, the set of feasible configurations for every data stream $(i, j) \in E_{dsp}$.

4.1.2 Computing Infrastructure Model

The computing infrastructure hosting DSP applications comprises a set of computing nodes (being them powerful servers in data centers, or resource-constrained devices at the edge), and the network resources that interconnect them. Computing and network resources can be represented as a labeled, fully connected, directed

[1] Operator processing speed depends on the actual software/hardware architecture where it is executed. To this end, we define the operator speed with respect to a reference implementation on a reference architecture.

graph $G_{res} = (V_{res}, E_{res})$. Vertices in V_{res} represent the distributed computing nodes, whereas the edges in E_{res} represent the *logical links* between nodes, which result by the underlying physical network paths and routing strategies, not modeled at our level of abstraction. We also model edges of the type (u, u), which capture local connectivity between operators placed in the same node u.

We characterize each node $u \in V_{res}$ by means of numerical attributes: Res_u, the amount of computing resources available at u; c_u, the monetary cost per unit of time associated with deploying an operator in the node; S_u^ϕ, the processing speedup with respect to a reference processor, when using configuration ϕ.

Each node $u \in V_{res}$ is also associated with a set of node *configurations*, Φ_u. The concept of node configuration is analogous to the concept of operator configuration, introduced above. We recall that a configuration ϕ is a vector $\phi = \left(v_{\rho_1}, v_{\rho_2}, \ldots, v_{\rho_N}\right)$, where $v_{\rho_i} \in V_{\rho_i}$ is a value associated with the RO ρ_i, and $\{\rho_1, \rho_2, \ldots, \rho_N\} \subseteq \Omega_{RO}$. The set of configurations available on u depends on the hardware/software capabilities of u; it can be defined as:

$$\Phi_u \subseteq \left\{\left(v_{\rho_1}, v_{\rho_2}, \ldots\right) : \forall v_{\rho_k} \in V_{\rho_k}^u, \forall \rho_k \in \Omega_{RO}^u\right\} \tag{2}$$

where $V_\rho^u \subseteq V_\rho$ is the set of values for the RO ρ supported by u, and $\Omega_{RO}^u \subseteq \Omega_{RO}$ is the subset of ROs characterizing the node u. Note that in the definition above the equality does not necessarily hold, as some combinations of values for different ROs may be unfeasible in practice.

Similar definitions apply to the communication links $(u, v) \in E_{res}$, each being associated with a set of configurations $\Phi_{(u,v)}$, defined as:

$$\Phi_{(u,v)} \subseteq \left\{\left(v_{\rho_1}, v_{\rho_2}, \ldots\right) : \forall v_{\rho_k} \in V_{\rho_k}^{(u,v)}, \forall \rho_k \in \Omega_{RO}^{(u,v)}\right\} \tag{3}$$

where $V_\rho^{(u,v)} \subseteq V_\rho$ is the set of values for the RO ρ supported by (u, v), and $\Omega_{RO}^{(u,v)} \subseteq \Omega_{RO}$ is the subset of ROs characterizing (u, v). Each link (u, v) and its configurations are also described by: $d_{(u,v)}^\phi$, the network delay between node u and v, when using network configuration ϕ; $B_{(u,v)}$, the network capacity available between u and v; $c_{(u,v)}$, the monetary cost per unit of data exchanged on the link; $\alpha_{(u,v)}^\phi$, a coefficient that captures any data transmission overhead incurred when using link configuration ϕ (e.g., data encryption can lead to sending extra information).

4.2 DSP Placement Model

The DSP placement problem consists in determining a suitable mapping between the DSP graph G_{dsp} and the resource graph G_{res}, as illustrated in Fig. 2. It is often the case that some operators, especially sources and sinks, are *pinned*, that is their placement is fixed. Without loss of generality, we assume that the other operators can be placed on any node of the computing infrastructure, provided that the application requirements for that operator can be satisfied when running in that node.

We can conveniently model the DSP placement problem with binary variables $x_{i,u}^{\phi}$, $i \in V_{dsp}$, $u \in V_{res}$, $\phi \in \Phi_u$. The variable $x_{i,u}^{\phi}$ has value 1 if operator i is deployed on node u using configuration ϕ, and zero otherwise. A correct placement must deploy an operator on one and only one computing node; this condition can be guaranteed requiring that $\sum_u \sum_\phi x_{i,u}^{\phi} = 1$, with $u \in V_{res}$, $i \in V_{dsp}$, $\phi \in \Phi_u$. For each pinned operator i_p, being $\bar{u}$ the node it must be deployed on, clearly we have that $x_{i_p,u}^{\phi} = 0$ for every node $u \neq \bar{u}$. Furthermore, under some configurations $\Phi_u^{\epsilon} \subseteq \Phi_u$ a node u becomes available for *exclusive use* of a single operator. We model the choice of exclusively acquiring nodes with additional binary variables w_u, $u \in V_{res}$, which have value 1 if the node u is exclusively used, and zero otherwise.

We also consider binary variables associated with links, namely $y_{(i,j),(u,v)}^{\phi}$, $(i, j) \in E_{dsp}$, $(u, v) \in E_{res}$, $\phi \in \Phi_{(u,v)}$, which denotes whether the data stream flowing from operator i to operator j traverses the network path from node u to node v, using network configuration ϕ.

For short, in the following we denote by $\boldsymbol{x}$ and $\boldsymbol{y}$ the placement vectors for nodes and edges, respectively, where $\boldsymbol{x} = \langle x_{i,u}^{\phi} \rangle$, $\forall i \in V_{dsp}$, $\forall u \in V_{res}$, $\forall \phi \in \Phi_u$ and $\boldsymbol{y} = \left\langle y_{(i,j),(u,v)}^{\phi} \right\rangle$, $\forall (i, j) \in E_{dsp}$, $\forall (u, v) \in E_{res}$, $\forall \phi \in \Phi_{(u,v)}$. Similarly, we define the exclusive use vector $\boldsymbol{w} = \langle w_u \rangle$, $\forall u \in V_{res}$.

Note that our model does not take into account operator replication, that is allocating multiple parallel replicas of operators to handle larger volumes of input data. The model we present can be easily extended to consider this scenario, either by tweaking the decision variables as in [14], or by introducing an intermediate DSP graph representation, which contains the replicated operators, and then computing the placement for this new graph. However, both these approaches would make the formulation less readable, and thus we do not investigate the replication problem in this chapter.

4.3 QoS and Cost Metrics

Placement decisions have a significant impact on both the achieved QoS and the associated monetary deployment cost. In this section, we introduce the metrics we are interested in optimizing, and will be used in our optimization problem formulation. Specifically, we are interested in optimizing the trade-offs between performance, security- and privacy-related requirements satisfaction, and deployment cost. To this end, in Sect. 4.3.1 we will consider application response time as the reference performance metric; in Sect. 4.3.2 we will show how requirement satisfaction metrics can be formulated; and in Sect. 4.3.3 we will define the deployment cost metrics we will use. Note that other metrics can be easily included in our optimization framework, e.g., availability or network-related metrics [14]. We do not show their formulation here because of space limitations.

4.3.1 Response Time

DSP applications are often used in latency-sensitive domains, where we are interested in processing newly incoming data as soon as possible. For this reason, a relevant performance metric for DSP application is *response time*. Given a source-to-sink path in the application DAG, we define response time as the time it takes for a data unit emitted by the source to reach the sink, and thus possibly producing a result/response.

In order to formalize the above definition, we consider a source-to-sink path $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_{N_\pi})$, $\pi_i \in V_{dsp}$. We define the application response time associated with the path $\boldsymbol{\pi}$ as:

$$R^{\boldsymbol{\pi}}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=\pi_1}^{\pi_{N_\pi}} R_i(\boldsymbol{x}) + \sum_{p=1}^{N_\pi - 1} D_{(\pi_p, \pi_{p+1})}(\boldsymbol{y}) \tag{4}$$

where the first term accounts for the processing time spent at each operator in $\boldsymbol{\pi}$, and the second term accounts for the network delay accumulated along the path. In presence of multiple source-to-sink paths, we may also be interested in the overall application response time $R(\boldsymbol{x}, \boldsymbol{y})$, which we define as the maximum response time among all the paths, i.e., $R(\boldsymbol{x}, \boldsymbol{y}) = \max_{\boldsymbol{\pi}} R^{\boldsymbol{\pi}}(\boldsymbol{x}, \boldsymbol{y})$.

The single operator response time in turn can be formulated as:

$$R_i(\boldsymbol{x}) = \sum_{u \in V_{res}} \sum_{\phi \in \Phi_u} R_i(\lambda_i, S_u^\phi) x_{i,u}^\phi \tag{5}$$

where $R_i(\lambda, S)$ is the operator response time, evaluated with respect to the current operator input rate λ_i, and processing speedup S. The total network delay along the path $\boldsymbol{\pi}$ is equal to:

$$D_{(i,j)}(\boldsymbol{y}) = \sum_{(u,v) \in E_{res}} \sum_{\phi \in \Phi_{(u,v)}} d_{(u,v)}^\phi y_{(i,j),(u,v)}^\phi \tag{6}$$

4.3.2 Security Requirements

Performance-related metrics are not sufficient to characterize the application QoS in highly distributed fog-like environments. Actually, it is necessary to take into account additional, non-functional application requirements, especially those related to security needs. To this end, in addition to the traditionally used response time metric, we consider a set of metrics that allow quantitatively reasoning about the satisfaction of application requirements. In particular, we will define metrics of the type $S_\alpha^\beta(\boldsymbol{x}, \boldsymbol{y})$, which measure the satisfaction level of (a subset of) the application requirements, with respect to a certain deployment configuration. All these metrics take value in [0, 1], with 1 indicating perfect matching of the application requirements, and 0 no satisfaction at all.

Given a placement $(\boldsymbol{x}, \boldsymbol{y})$, we define an application-level satisfaction metric $S(\boldsymbol{x}, \boldsymbol{y})$, which considers the requirements from every RC:

$$S(\boldsymbol{x}, \boldsymbol{y}) = \prod_{\omega \in \Omega_{RC}} S_\omega(\boldsymbol{x}, \boldsymbol{y}) \tag{7}$$

Then, we define the satisfaction of application requirements related to RC $\omega \in \Omega_{RC}$ as:

$$S_\omega(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i \in V_{dsp}} S_\omega^i(\boldsymbol{x}) \cdot \prod_{(i,j) \in E_{dsp}} S_\omega^{(i,j)}(\boldsymbol{y}) \tag{8}$$

where $S_\omega^i(\boldsymbol{x})$ measures the requirements satisfaction for operator i, and $S_\omega^{(i,j)}(\boldsymbol{y})$ for data stream (i, j), with respect to ω. That is, the satisfaction of the requirements in ω implies meeting the requirements of all the operators *and* all the streams. In turn, evaluating the requirements satisfaction for an operator (or, equivalently, a stream) with respect to the current placement $\boldsymbol{x}$, means evaluating how much its current configuration matches its requirements. Formally, we let $\phi(\boldsymbol{x}, i)$ denote the node configuration in use by operator i under placement $\boldsymbol{x}$, i.e., $\phi(\boldsymbol{x}, i)$ identifies the unique ϕ such that, for any node $u \in V_{res}$, $x_{i,u}^\phi = 1$. We have:

$$S_\omega^i(\boldsymbol{x}) = S_\omega^i(\phi(\boldsymbol{x}, i)) \tag{9}$$

Henceforth, for each operator i, we need to evaluate $S_\omega^i(\phi)$ in order to assess how much a configuration ϕ satisfies its requirements in ω. Recalling that $\Omega_{RO}^{\omega,i}$ denotes the set of ROs in ω that characterize the deployment of i, which are in conjunction with each other, we get:

$$S_\omega^i(\phi) = \prod_{\rho \in \Omega_{RO}^{\omega,i}} q_{\phi_\rho}^{i,\rho} \tag{10}$$

where $q_{\phi_\rho}^{i,\rho} \in [0, 1]$ is the preference value assigned by operator i to ϕ_ρ, and ϕ_ρ is the value that characterizes ρ in configuration ϕ.[2]

[2] We note that the vector ϕ possibly specifies a value ϕ_ρ for a subset of ROs $\tilde{\Omega}_{RO} \subseteq \Omega_{RO}$. Thus, with a slight abuse of notation, we assume $q_{\phi_\rho}^{i,\rho} = 0, \forall \rho \notin \tilde{\Omega}_{RO}$, i.e., a configuration ϕ cannot satisfy requirements for any not specified RO.

Equivalent metrics can be formulated for data streams in a similar way. We have:

$$S_{\omega}^{(i,j)}(\boldsymbol{y}) = S_{\omega}^{(i,j)}(\phi(\boldsymbol{y}, (i, j))) \tag{11}$$

where $\phi(\boldsymbol{y}, (i, j))$ is the unique ϕ such that, for any link $(u, v) \in E_{res}$, $y_{(i,j),(u,v)}^{\phi} = 1$. We can thus evaluate the requirement satisfaction for (i, j) under configuration ϕ:

$$S_{\omega}^{(i,j)}(\phi) = \prod_{\rho \in \Omega_{RO}^{\omega,(i,j)}} q_{\phi_\rho}^{(i,j),\rho} \tag{12}$$

4.3.3 Deployment Cost

We model the monetary deployment cost associated with the usage of computing and network resources. We assume a typical pay-as-you-go cost model, where the monetary cost for each resource is proportional to its usage. In this scenario, we define the total deployment cost for the application, per unit of time, as $C(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w})$. It accounts for both the cost of the computing resources and the network usage, as follows:

$$C(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w}) = \sum_{u \in V_{res}} C_u(\boldsymbol{x}, \boldsymbol{w}) + \sum_{(u,v) \in E_{res}} C_{(u,v)}(\boldsymbol{y}) \tag{13}$$

The computing nodes cost $C_u(\boldsymbol{x}, \boldsymbol{w})$ in turn can be formulated as:

$$C_u(\boldsymbol{x}, \boldsymbol{w}) = \sum_{\phi \in \Phi_u \setminus \Phi_u^{\epsilon}} \sum_{i \in V_{dsp}} Res_i c_u x_{i,u}^{\phi} + Res_u c_u w_u \tag{14}$$

where the first term accounts for the cost paid when the node is not exclusively acquired by an operator, which is proportional to Res_i, the amount of resources allocated to each operator i; the second term instead accounts for the cost paid when the node is exclusively used by an operator, which is equivalent to paying for all the resources provided by u, Res_u.

The network usage cost is defined as:

$$C_{(u,v)}(\boldsymbol{y}) = c_{(u,v)} N_{(u,v)}(\boldsymbol{y}) \tag{15}$$

with $N_{(u,v)}(\boldsymbol{y})$ representing the amount of data exchanged through the link (u, v), which can be computed as follows:

$$N_{(u,v)}(\boldsymbol{y}) = \sum_{\phi \in \Phi_{(u,v)}} \sum_{(i,j) \in E_{dsp}} b_{(i,j)} \lambda_{(i,j)} \alpha_{(u,v)}^{\phi} y_{(i,j),(u,v)}^{\phi} \tag{16}$$

where $b_{(i,j)}\lambda_{(i,j)}$ represents the data rate of the stream (i, j), and $\alpha^{\phi}_{(u,v)}$ is a coefficient that captures any data transmission overhead incurred when using link configuration ϕ.

5 Security-Aware DSP Placement Problem Formulation

When determining the placement of a DSP application, we aim at optimizing a QoS-based function $F(\boldsymbol{x}, \boldsymbol{y})$ that involves one or more of the metrics presented in the previous section. Moreover, we are often expected to additionally satisfy one or more constraints, revolving around the same metrics of interest. For example, a fixed monetary budget may be allocated for running the applications; performance-related Service Level Objectives (SLOs) may have to be met; security requirements of critical operators may have to be necessarily matched in order for the application to operate.

In this work, relying on the QoS and cost metrics described above, we consider three types of constraints, respectively related to the deployment cost, application performance, and requirements satisfaction.

Deployment Cost Given a defined available budget C_{max}, the total application deployment cost must not exceed C_{max}.

Performance We assume performance-related SLO to be defined for the application, expressed in terms of application response time. In particular, an upper bound R^{π}_{max} is defined for each source-to-sink path $\boldsymbol{\pi}$. Distinct paths indeed possibly carry out processing tasks that are more or less latency-critical, and thus can be subject to different SLOs.

Security-Related Requirements Given a specification of the application Requirements Forest, and the requirement satisfaction metrics defined in the previous section, several constraints can be introduced in the deployment optimization problem, which allow to model both "hard" and "soft" security requirements. The generic requirement satisfaction constraint has the form:

$$S^{\beta}_{\alpha}(\cdot) \geq S^{\beta}_{\alpha,min} \tag{17}$$

By replacing $S^{\beta}_{\alpha}(\cdot)$ with the appropriate concrete metric, the constraint can be applied to a specific subset of the requirements forest. For example, by using $S_{\omega}(\boldsymbol{x}, \boldsymbol{y})$, we can formulate a constraint on the satisfaction of the requirements in the RC $\omega \in \Omega_{RC}$; using $S^{i}_{\rho}(\phi(\boldsymbol{x}, i))$, we can formulate a constraint associated with a specific operator $i \in V_{dsp}$, and a single RO $\rho \in \Omega_{RO}$; instead, using $S(\boldsymbol{x}, \boldsymbol{y})$, the constraint applies to the whole application requirement forest.

Furthermore, by properly setting the lower bound $S^{\beta}_{\alpha,min}$, we can model different kinds of requirements. If $S^{\beta}_{\alpha,min} = 0$ and strict inequality is used, the constraint only prohibits using deployment solutions where configurations not accepted by the application are used (e.g., an operator requiring a Linux-based OS cannot be deployed in a Windows-based node); if $S^{\beta}_{\alpha,min} = 1$ we get a hard constraint, where every single involved requirement must be completely satisfied; if $S^{\beta}_{\alpha,min} \in (0, 1)$ we get a soft constraint, where only a minimum level of satisfaction is needed, based on the preference values assigned to the various configurations.

5.1 Problem Formulation

We formulate the Security-aware DSP Placement (SDP) problem as an Integer Linear Programming (ILP) model as follows:

$$\min_{\boldsymbol{x},\boldsymbol{y}} F(\boldsymbol{x}, \boldsymbol{y})$$

$$\textbf{subjectto}:$$

$$R^{\pi}_{max} \geq R^{\pi}(\boldsymbol{x}, \boldsymbol{y}) \qquad \forall \pi \in \Pi_{dsp} \tag{18}$$

$$C_{max} \geq C(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w}) \tag{19}$$

$$\tilde{S}^{\beta}_{\alpha,min} \leq \tilde{S}^{\beta}_{\alpha}(\boldsymbol{x}, \boldsymbol{y}) \tag{20}$$

$$B_{(u,v)} \geq N_{(u,v)}(\boldsymbol{y}) \qquad \forall (u, v) \in E_{res} \tag{21}$$

$$Res_u \geq \sum_{i \in V_{dsp}} \sum_{\phi \in \Phi_u} Res_i x^{\phi}_{i,u} \qquad \forall u \in V_{res} \tag{22}$$

$$1 = \sum_{u \in V_{res}} \sum_{\phi \in \Phi_u} x^{\phi}_{i,u} \qquad \forall i \in V_{dsp} \tag{23}$$

$$\sum_{\phi \in \Phi_u} x^{\phi}_{i,u} = \sum_{v \in V_{res}} \sum_{\phi \in \Phi_{(u,v)}} y^{\phi}_{(i,j),(u,v)} \qquad \forall (i,j) \in E_{dsp},\ u \in V_{res} \tag{24}$$

$$\sum_{\phi \in \Phi_v} x^{\phi}_{j,v} = \sum_{u \in V_{res}} \sum_{\phi \in \Phi_{(u,v)}} y^{\phi}_{(i,j),(u,v)} \qquad \forall (i,j) \in E_{dsp},\ v \in V_{res} \tag{25}$$

$$w_u \geq \sum_{i \in V_{dsp}} \sum_{\phi \in \Phi^{\epsilon}_u} x^{\phi}_{i,u} \qquad \forall u \in V_{res} \tag{26}$$

$$(1 - w_u)M \geq \sum_{i \in V_{dsp}} \sum_{\phi \in \Phi_u \setminus \Phi^{\epsilon}_u} x^{\phi}_{i,u} \qquad \forall u \in V_{res} \tag{27}$$

$$w_u \in \{0, 1\} \qquad \forall u \in V_{res}$$

$$x^{\phi}_{i,u} \in \{0, 1\} \qquad \forall i \in V_{dsp},\ u \in V_{res},\ \phi \in \Phi_u$$

$$y^{\phi}_{(i,j),(u,v)} \in \{0,1\} \qquad \begin{array}{c} \forall (i,j) \in E_{dsp}, \\ (u,v) \in E_{res}, \\ \phi \in \Phi_{(u,v)} \end{array}$$

where $M \gg 1$ is a large constant, and $\tilde{S}(\cdot) = \log S(\cdot)$ is used in order to obtain linear expressions for the requirements satisfaction metrics. Constraints (18)–(20) model the QoS- and cost-related bounds described above, where—as explained—(20) may actually be replaced by several constraints involving different security requirements. Constraints (21) and (22) are capacity constraints, modeling, respectively, the limited capacity of network links, and the limited amount of computational resources available at nodes. Equation (23) reflects the fact that a single node and a single configuration must be chosen for each operator. Equations (24) and (25) are flow conservation constraints, which model the logical AND relationship between placement variables. Constraints (26) and (27) model the relationship between exclusive use variables and placement configurations.

It is easy to realise that SDP is a NP-hard problem. To this end, it suffices to observe that SDP is a generalization of the optimal DSP placement problem presented in [13], which has been shown to be NP-hard.

5.2 Example

We formulate the SDP problem for the smart health application presented in the previous sections. The requirements for this application, described in Sect. 3, were reported in Fig. 5. We further assume that a fixed budget C_{max} is allocated for deploying the application, and the application is expected to meet SLOs formulated in terms of response time. Specifically, denoting as $\boldsymbol{\pi}_1$ the operators path from the source to $sink_1$, we assume $\boldsymbol{\pi_1}$ to have strict latency requirements, and set the maximum response time along the path, $R^{\boldsymbol{\pi_1}}_{max}$, to 10 ms. For the other path, $\boldsymbol{\pi_2}$, we set $R^{\boldsymbol{\pi_2}}_{max} = 100$ ms. Moreover, as we want none of the application requirements to be ignored, we require $S(\boldsymbol{x}, \boldsymbol{y})$ to be strictly greater than zero.

We consider different scenarios for formulating the SDP problem, as follows:

- *Scenario A*: we solve SDP maximizing the satisfaction of application requirements, i.e., $F(\boldsymbol{x}, \boldsymbol{y}) = \tilde{S}(\boldsymbol{x}, \boldsymbol{y})$;
- *Scenario B*: we solve SDP minimizing the worst-case application response time, i.e., $F(\boldsymbol{x}, \boldsymbol{y}) = \max_{\boldsymbol{\pi} \in \boldsymbol{\Pi}_{dsp}} R_{\boldsymbol{\pi}}(\boldsymbol{x}, \boldsymbol{y})$. We also consider three cases, considering an additional constraint on overall requirements satisfaction: (i) $S(\boldsymbol{x}, \boldsymbol{y}) > 0$; (ii) $S(\boldsymbol{x}, \boldsymbol{y}) \geq 0.9$; (iii) $S(\boldsymbol{x}, \boldsymbol{y}) \geq 0.99$.

The computing infrastructure we consider for deployment is depicted in Fig. 6. The infrastructure is composed of 15 geographically distributed computing nodes: 3 edge nodes, 4 fog nodes distributed across two micro-data centers, and 8 cloud nodes distributed across two data centers. We assume that the application data source is pinned on the first edge node. The operators and the sinks can be freely

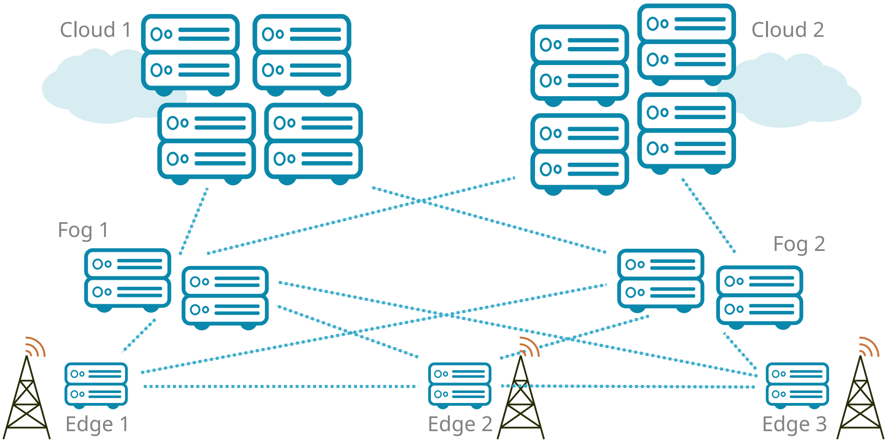


Fig. 6 Illustration of the computing infrastructure used in the example, comprising edge nodes, fog micro-data centers, and cloud data centers

placed in any node of the infrastructure. Compared to a reference cloud node, processing at the fog nodes is 10% slower, and 20% slower at the edge. Conversely, cloud nodes are cheaper than those in the fog, which in turn are cheaper than edge nodes. We assume network delay to be negligible within the same geographical region (i.e., the same (micro-)data center). We further assume that fog and cloud nodes support operator execution either as bare processes, or within containers or within VMs, whereas operators cannot be deployed in edge nodes using VMs. Moreover, we assume edge nodes cannot be acquired exclusively by a single operator in this scenario.

Results in Scenario A In this scenario, SDP aims at maximizing the security requirements satisfaction. To illustrate the results, we solve SDP considering different choices for the monetary budget C_{max}, namely 5, 6, 7, and 10 \$/h. The optimization results in this scenario are reported in Table 1. In Fig. 7, we show the deployment computed by SDP for $C_{max} \in \{5, 10\}$ \$/h. In all the considered experiments, traffic encryption is enforced for all the data streams in the application. The minimum budget $C_{max} = 5$ \$/h leads to a requirements satisfaction degree equal to 0.9. As illustrated in Fig. 7a, in this case fog and cloud nodes are used for deploying application operators (except for the pinned data source), with op_2 running in an exclusively acquired node. Conversely, op_1 is deployed in a multi-tenant node, leading to partial requirements satisfaction. When the cost budget is raised to 10 \$/h, we can note that (i) op_1 is deployed in a dedicated node, completely satisfying its isolation requirement, (ii) the application path to $sink_2$ is deployed in the fog, without relying on any cloud node, and (iii) the other application path is deployed across 3 geographical regions instead of 4 as in the previous case. The total application response time along the two paths is thus reduced, respectively,

Table 1 Optimization results in *Scenario A* of the example

C_{max} (\$/h)	C (\$/h)	R^{π_1} (ms)	R^{π_2} (ms)	S
5	4.6	9.1	6.0	0.9
6	5.9	9.1	12.0	1.0
7	6.5	9.1	9.0	1.0
10	9.7	6.1	4.0	1.0

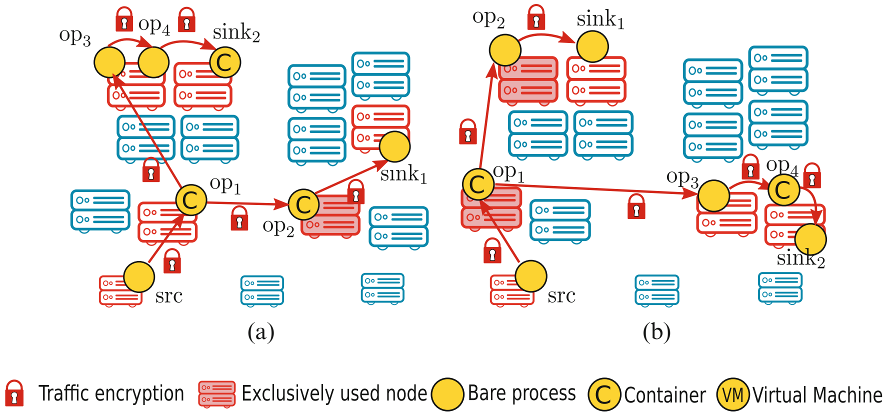


Fig. 7 Deployment computed by SDP in example Scenario A. (**a**) $C_{max} = 5$ \$/h. (**b**) $C_{max} = 10$ \$/h

from 9 ms to 6 ms and from 6 ms to 4 ms. However, the deployment cost is doubled, from 4.6 \$/h to 9.7 \$/h.

In the other configurations, instead, with slightly larger allocations for C_{max}, SDP is able to perfectly match application requirements. In all the cases, the response time along both the paths is within the SLO bound.

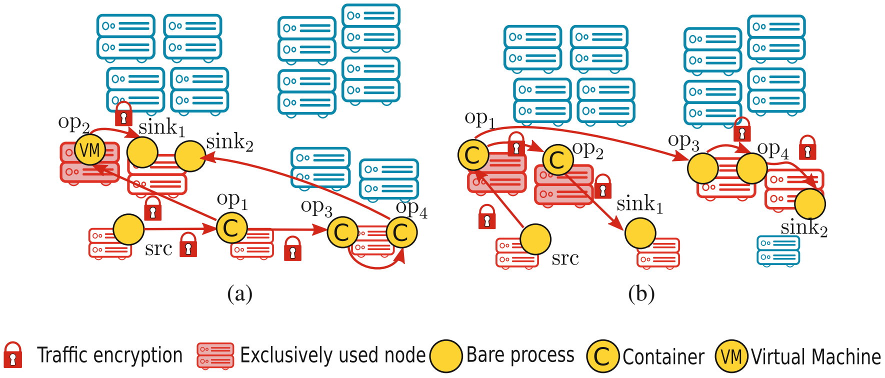
Results in Scenario B In this scenario we aim at minimizing the application response time, while requiring a minimum level of requirements satisfaction, S_{min}. We again consider different values for the maximum cost $C_{max} \in \{5, 6, 7, 10\}$, and solve SDP varying the requirements satisfaction constraint. In Table 2 we report the optimization results, while in Fig. 8 we show the deployment computed by SDP for two illustrative cases.

Requiring $S(\boldsymbol{x}, \boldsymbol{y}) > 0$ means that only configurations with non-zero requirements satisfaction can be adopted. Whatever the monetary budget, in this case SDP computes placement solutions where requirements satisfaction is rather low. Figure 8a shows the solution for $C_{max} = 5$ \$/h. In order to minimize latency, differently from Scenario A, for this scenario SDP does not use cloud nodes. Traffic encryption is only used on the “critical” path towards $sink_1$, and only op_2 is deployed in a dedicated node, as it does not admit other configurations.

When we require $S(\boldsymbol{x}, \boldsymbol{y}) \geq 0.9$, SDP computes solutions characterized by higher requirements satisfaction (0.9 in all the considered settings), with negligible

Table 2 Optimization results in *Scenario B* of the example. The case where S_{min} is set to 0.99 is not feasible when a budget of 5 $/h is allocated

C_{max} ($/h)	S_{min}	C ($/h)	R^{π_1} (ms)	R^{π_2} (ms)	S
5	> 0	4.6	6.1	6.0	0.11
6	> 0	5.2	1.1	6.0	0.23
7	> 0	5.2	1.1	6.0	0.45
10	> 0	9.4	1.1	1.0	0.23
5	0.90	4.6	6.1	6.0	0.90
6	0.90	5.2	1.1	6.0	0.90
7	0.90	5.2	1.1	6.0	0.90
10	0.90	9.4	1.1	1.0	0.90
5	0.99	-	-	-	-
6	0.99	5.9	6.1	6.0	1.00
7	0.99	6.5	4.1	6.0	1.00
10	0.99	9.1	2.1	4.0	1.00



(a) (b)

Fig. 8 Deployment computed by SDP in example Scenario B

impact on application response time and deployment cost. Interestingly, when we require $S(\boldsymbol{x}, \boldsymbol{y}) \geq 0.99$, SDP is not able to find a feasible solution if the monetary budget is 5 $/h. Indeed, it would need to exclusively acquire more than one computing node, and that is expensive. With a higher budget instead SDP is able to perfectly match the application requirements. We note that this happens at the cost of slightly higher response time, especially on the critical path $\boldsymbol{\pi}_1$. Figure 8b shows the deployment determined when $C_{max} = 10$ $/h. Again, "far" cloud nodes are not used in this scenario. In this setting, traffic encryption is enabled for all the data streams, and an additional fog node is exclusively acquired for deploying op_1 to satisfy the security requirements.

6 Discussion

State-of-the-art solutions for the DSP placement problem allow to effectively optimize several QoS metrics, taking into account both functional and non-functional characterizations of the application operators and the underlying computing infrastructure. However, when we move applications out of the traditionally used data centers, we are forced to look at security, privacy, and data integrity concerns as major issues. Unfortunately, existing techniques often completely neglect these aspects.

SDP aims at overcoming this limitation by reserving a primary role to application deployment requirements. The forest-based approach we presented for specifying application requirements allows to easily integrate satisfaction metrics into the placement optimization problem; moreover, it provides large flexibility for formalizing application needs. In particular, compared to similar works in the literature, our approach allows—when needed—to apply different policies to different types of requirements (e.g., to multiple RCs), and to different operators/data streams, instead of necessarily collapsing "security" to a single numerical indicator. It is worth observing that the requirements formalism and the associated metrics presented in this chapter might not perfectly suit every application domain. Nevertheless, SDP can be easily adapted to work with a different toolbox for quantitative modeling of security aspects (e.g., a probabilistic model like that presented in [25]).

The concept of operator (data stream) and node (link) configurations we introduced in SDP provides additional degrees of freedom in the placement optimization. Compared to previous work on the topic, SDP goes beyond simply determining a mapping from operators to computing nodes, by also identifying the specific software configuration to be adopted for each operator. The integration of this configuration optimization approach with existing software orchestration platforms will be subject of further investigations, in order to have a system capable of automatically choosing and *applying* necessary configuration for the deployed application.

As noted above, some possible extensions of the presented problem formulation were intentionally not tackled in this work, and deferred to future research. First of all, the formulation can be easily generalized to take into account the operator replication problem, e.g., adopting the approach used in [14]. Moreover, we have not specifically covered here the issues related to run-time deployment adaptation, focusing instead on the initial application placement. Nonetheless, adaptation overhead metrics can be readily introduced in SDP, e.g., following our previous work [12].

7 Conclusion

In this chapter, we looked at how security aspects impact the placement problem for DSP applications. The recent trend of shifting data analytics services from traditionally used cloud data centers to fog computing environments, in order to reduce network latency between applications and data sources, forces us to deal with a broader range of security and privacy issues.

We presented an approach, based on a forest of AND-OR trees, for specifying additional non-functional application requirements, which are hardily captured by existing techniques for placement optimization. Relying on this formalism, we also defined a set of metrics that allow to quantitatively reason about requirements satisfaction, especially as regards security aspects. We included these metrics in the ILP-based SDP problem, which determines the optimal DSP application placement according to several QoS constraints. By means of an illustrative case study of a smart health application in the fog, we provided insights about the trade-offs between performance, cost, and security computed by SDP. Our approach provides great flexibility for specifying requirements, as well as optimization objectives and constraints. We pointed out some open research directions, especially as regards the integration of SDP with existing DSP frameworks. As future work, we plan to investigate this direction, by complementing our modeling effort with experimental validations, where concrete cybersecurity issues must be fitted within SDP.

References

1. Abadi, D.J., Ahmad, Y., Balazinska, M., Çetintemel, U., et al.: The design of the Borealis stream processing engine. In: Proc. CIDR ’05, pp. 277–289 (2005)
2. Agbo, C.C., Mahmoud, Q.H., Mikael Eklund, J.: A scalable patient monitoring system using Apache Storm. In: Proc. 2018 IEEE Canadian Conf. on Electrical Computer Engineering, pp. 1–6, CCECE ’18 (2018)
3. Anh, D.T.T., Datta, A.: Streamforce: Outsourcing access control enforcement for stream data to the clouds. In: Proc. ACM CODASPY ’14, pp. 13–24 (2014)
4. Aniello, L., Baldoni, R., Querzoni, L.: Adaptive online scheduling in Storm. In: Proc. ACM DEBS ’13, pp. 207–218 (2013)
5. Arkian, H.R., Diyanat, A., Pourkhalili, A.: MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications. J. Parallel Distrib. Comput. **82**, 152–165 (2017)
6. Backman, N., Fonseca, R., Çetintemel, U.: Managing parallelism for stream processing in the cloud. In: Proc. HotCDP ’12, pp. 1:1–1:5. ACM (2012)
7. Bellendorf, J., Mann, Z.A.: Classification of optimization problems in fog computing. Future Gener. Comput. Syst. **107**, 158–176 (2020)
8. Brogi, A., Forti, S., Guerrero, C., Lera, I.: How to place your apps in the fog: State of the art and open challenges. Softw. Pract. Exp. (2019)
9. Burkhalter, L., Hithnawi, A., Viand, A., Shafagh, H., Ratnasamy, S.: TimeCrypt: Encrypted data stream processing at scale with cryptographic access control. In: Proc. USENIX NSDI ’20, pp. 835–850 (Feb 2020)

10. Cao, J., Carminati, B., Ferrari, E., Tan, K.L.: ACStream: Enforcing access control over data streams. In: Proc. IEEE ICDE '09, pp. 1495–1498 (2009)
11. Cardellini, V., Lo Presti, F., Nardelli, M., Rossi, F.: Self-adaptive container deployment in the fog: A survey. In: Proc. ALGOCLOUD '19. LNCS. Springer (2020)
12. Cardellini, V., Lo Presti, F., Nardelli, M., Russo Russo, G.: Optimal operator deployment and replication for elastic distributed data stream processing. Concurr. Comput. Pract. Exp. **30**(9) (2018)
13. Cardellini, V., Grassi, V., Lo Presti, F., Nardelli, M.: Optimal operator placement for distributed stream processing applications. In: Proc. ACM DEBS '16, pp. 69–80 (2016)
14. Cardellini, V., Grassi, V., Lo Presti, F., Nardelli, M.: Optimal operator replication and placement for distributed stream processing systems. ACM SIGMETRICS Perfom. Eval. Rev. **44**(4), 11–22 (May 2017)
15. Carminati, B., Ferrari, E., Cao, J., Tan, K.L.: A framework to enforce access control over data streams. ACM Trans. Inf. Syst. Secur. **13**(3) (Jul 2010)
16. Carminati, B., Ferrari, E., Tan, K.: Enforcing access control over data streams. In: Proc. ACM SACMAT '07, pp. 21–30 (2007)
17. Chandramouli, B., Goldstein, J., Barga, R., Riedewald, M., Santos, I.: Accurate latency estimation in a distributed event processing system. In: Proc. IEEE ICDE '11, pp. 255–266 (2011)
18. Chaturvedi, S., Simmhan, Y.: Toward resilient stream processing on clouds using moving target defense. In: Proc. IEEE ISORC '19, pp. 134–142 (2019)
19. Chatzistergiou, A., Viglas, S.D.: Fast heuristics for near-optimal task allocation in data stream processing over clusters. In: Proc. ACM CIKM '14, pp. 1579–1588 (2014)
20. de Assunção, M.D., da Silva Veith, A., Buyya, R.: Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. J. Netw. Comput. Appl. **103**, 1–17 (2018)
21. Eidenbenz, R., Locher, T.: Task allocation for distributed stream processing. In: Proc. IEEE INFOCOM '16 (2016)
22. Eskandari, L., Mair, J., Huang, Z., Eyers, D.: T3-Scheduler: A topology and traffic aware two-level scheduler for stream processing systems in a heterogeneous cluster. Future Gener. Comput. Syst. **89**, 617–632 (2018)
23. Fischer, L., Scharrenbach, T., Bernstein, A.: Scalable linked data stream processing via network-aware workload scheduling. In: Proc. 9th Int'l Workshop Scalable Semantic Web Knowledge Base Systems (2013)
24. Fisher, R., Hancke, G.: DTLS for lightweight secure data streaming in the Internet of Things. In: Proc. 2014 9th Int'l Conf. on P2P, Parallel, Grid, Cloud and Internet Computing, pp. 585–590 (Nov 2014)
25. Forti, S., Ferrari, G.L., Brogi, A.: Secure cloud-edge deployments, with trust. Future Gener. Comput. Syst. **102**, 775–788 (2020)
26. Gedik, B., Özsema, H., Öztürk, O.: Pipelined fission for stream programs with dynamic selectivity and partitioned state. J. Parallel Distrib. Comput. **96**, 106–120 (2016)
27. Ghaderi, J., Shakkottai, S., Srikant, R.: Scheduling storms and streams in the cloud. ACM Trans. Model. Perform. Eval. Comput. Syst. **1**(4), 14:1–14:28 (2016)
28. Ghosh, R., Simmhan, Y.: Distributed scheduling of event analytics across edge and cloud. ACM Trans. Cyber Phys. Syst. **2**(4) (Jul 2018)
29. Gu, L., Zeng, D., Guo, S., Xiang, Y., Hu, J.: A general communication cost optimization framework for big data stream processing in geo-distributed data centers. IEEE Trans. Comput. **65**(1), 19–29 (2016)
30. Havet, A., Pires, R., Felber, P., Pasin, M., Rouvoy, R., Schiavoni, V.: SecureStreams: A reactive middleware framework for secure data stream processing. In: Proc. ACM DEBS '17, pp. 124–133 (2017)
31. Janßen, G., Verbitskiy, I., Renner, T., Thamsen, L.: Scheduling stream processing tasks on geo-distributed heterogeneous resources. In: Proc. IEEE Big Data '18, pp. 5159–5164 (2018)

32. Jiang, J., Zhang, Z., Cui, B., Tong, Y., Xu, N.: StroMAX: Partitioning-based scheduler for real-time stream processing system. In: Proc. DASFAA '17, pp. 269–288. Springer (2017)
33. Khare, S., Sun, H., Gascon-Samson, J., Zhang, K., Gokhale, A., Barve, Y., Bhattacharjee, A., Koutsoukos, X.: Linearize, predict and place: Minimizing the makespan for edge-based stream processing of directed acyclic graphs. In: Proc. ACM/IEEE SEC '19, pp. 1–14 (2019)
34. Lakshmanan, G.T., Li, Y., Strom, R.: Placement of replicated tasks for distributed stream processing systems. In: Proc. ACM DEBS '10, pp. 128–139 (2010)
35. Li, J., Deshpande, A., Khuller, S.: Minimizing communication cost in distributed multi-query processing. In: Proc. IEEE ICDE '09, pp. 772–783 (2009)
36. Li, T., Tang, J., Xu, J.: A predictive scheduling framework for fast and distributed stream data processing. In: Proc. 2015 IEEE Int'l Conf. on Big Data, pp. 333–338 (2015)
37. Lindner, W., Meier, J.: Securing the Borealis data stream engine. In: Proc. 10th Int'l Database Engineering and Applications Symp., pp. 137–147. IDEAS '06 (2006)
38. Loukopoulos, T., Tziritas, N., Koziri, M., Stamoulis, G., Khan, S.U.: A Pareto-efficient algorithm for data stream processing at network edges. In: Proc. IEEE CloudCom '18, pp. 159–162 (2018)
39. Luna Garcia, J., Langenberg, R., Suri, N.: Benchmarking cloud security level agreements using quantitative policy trees. In: Proc. 2012 ACM Workshop on Cloud Computing Security, pp. 103–112. CCSW '12 (2012)
40. Nardelli, M., Cardellini, V., Grassi, V., Lo Presti, F.: Efficient operator placement for distributed data stream processing applications. IEEE Trans. Parallel Distrib. Syst. **30**(8), 1753–1767 (2019)
41. Nehme, R.V., Lim, H., Bertino, E.: FENCE: continuous access control enforcement in dynamic data stream environments. In: Proc. IEEE ICDE '10, pp. 940–943 (2010)
42. Nehme, R.V., Rundensteiner, E.A., Bertino, E.: A security punctuation framework for enforcing access control on streaming data. In: Proc. IEEE ICDE '08, pp. 406–415 (2008)
43. Ng, W.S., Wu, H., Wu, W., Xiang, S., Tan, K.: Privacy preservation in streaming data collection. In: Proc. IEEE ICPADS '12, pp. 810–815 (Dec 2012)
44. O'Keeffe, D., Salonidis, T., Pietzuch, P.: Frontier: Resilient edge processing for the Internet of Things. Proc. VLDB Endow. **11**(10), 1178–1191 (Jun 2018)
45. Park, H., Zhai, S., Lu, L., Lin, F.X.: Streambox-TZ: Secure stream analytics at the edge with trustzone. In: Proc. USENIX ATC '19, pp. 537–554 (2019)
46. Peng, B., Hosseini, M., Hong, Z., Farivar, R., et al.: R-Storm: Resource-aware scheduling in Storm. In: Proc. Middleware '15, pp. 149–161. ACM (2015)
47. Peng, Q., Xia, Y., Wang, Y., Wu, C., Luo, X., Lee, J.: Joint operator scaling and placement for distributed stream processing applications in edge computing. In: Proc. ICSOC '19, pp. 461–476. LNCS. Springer (2019)
48. Pietzuch, P., Ledlie, J., Shneidman, J., Roussopoulos, M., et al.: Network-aware operator placement for stream-processing systems. In: Proc. IEEE ICDE '06 (2006)
49. Quoc, D.L., Beck, M., Bhatotia, P., Chen, R., Fetzer, C., Strufe, T.: PrivApprox: Privacy-preserving stream analytics. In: Proc. USENIX ATC '17, pp. 659–672 (Jul 2017)
50. Rizou, S., Durr, F., Rothermel, K.: Solving the multi-operator placement problem in large-scale operator networks. In: Proc. ICCCN '10, pp. 1–6 (2010)
51. Röger, H., Mayer, R.: A comprehensive survey on parallelization and elasticity in stream processing. ACM Comput. Surv. **52**(2), 36:1–36:37 (2019)
52. Rychly, M., Koda, P., Pavel: Scheduling decisions in stream processing on heterogeneous clusters. In: Proc. 8th Int'l Conf. Complex, Intelligent and Software Intensive Systems (2014)
53. Sajjad, H.P., Danniswara, K., Al-Shishtawy, A., Vlassov, V.: SpanEdge: Towards unifying stream processing over central and near-the-edge data centers. In: Proc. IEEE/ACM SEC '16, pp. 168–178 (2016)
54. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. Computer **29**(2), 38–47 (1996)
55. Sandhu, R.S., Samarati, P.: Access control: Principle and practice. IEEE Commun. Mag. **32**(9), 40–48 (1994)

56. Satyanarayanan, M., Klas, G., Silva, M., Mangiante, S.: The seminal role of edge-native applications. In: Proc. IEEE EDGE '19, pp. 33–40 (2019)
57. Schilling, B., Koldehofe, B., Rothermel, K., Ramachandran, U.: Access policy consolidation for event processing systems. In: Proc. NetSys '13, pp. 92–101. IEEE Computer Society (2013)
58. Sicari, S., Rizzardi, A., Grieco, L., Coen-Porisini, A.: Security, privacy and trust in Internet of Things: The road ahead. Comput. Netw. **76**, 146–164 (2015)
59. da Silva Veith, A., de Assunção, M.D., Lefèvre, L.: Latency-aware placement of data stream analytics on edge computing. In: Proc. ICSOC '18, pp. 215–229. LNCS. Springer (2018)
60. Smirnov, P., Melnik, M., Nasonov, D.: Performance-aware scheduling of streaming applications using genetic algorithm. Procedia Comput. Sci. **108**, 2240–2249 (2017)
61. Stanoi, I., Mihaila, G., Palpanas, T., Lang, C.: WhiteWater: Distributed processing of fast streams. IEEE Trans. Softw. Eng. **19**(9), 1214–1226 (2007)
62. Starks, F., Goebel, V., Kristiansen, S., Plagemann, T.: Mobile distributed complex event processing – Ubi sumus? Quo vadimus? In: Mobile Big Data: A Roadmap from Models to Technologies, pp. 147–180. Springer (2018)
63. Thoma, C., Labrinidis, A., Lee, A.J.: Automated operator placement in distributed data stream management systems subject to user constraints. In: Proc. IEEE ICDEW '14, pp. 310–316 (2014)
64. Thoma, C., Lee, A.J., Labrinidis, A.: PolyStream: Cryptographically enforced access controls for outsourced data stream processing. In: Proc. ACM SACMAT '16, pp. 227–238 (2016)
65. Tian, L., Chandy, K.M.: Resource allocation in streaming environments. In: Proc. 7th IEEE/ACM Int'l Conf. Grid Computing, pp. 270–277 (2006)
66. Xu, J., Chen, Z., Tang, J., Su, S.: T-Storm: Traffic-aware online scheduling in Storm. In: Proc. IEEE ICDCS '14, pp. 535–544 (2014)
67. Zhou, Y., Ooi, B.C., Tan, K.L., Wu, J.: Efficient dynamic operator placement in a locally distributed continuous query system. In: On the Move to Meaningful Internet Systems 2006, LNCS, vol. 4275, pp. 54–71. Springer (2006)
68. Zhuang, R., DeLoach, S.A., Ou, X.: Towards a theory of moving target defense. In: Proc. 1st ACM Workshop on Moving Target Defense, pp. 31–40. MTD '14 (2014)