Optimal Placement of Stream Processing Operators in the Fog

Thomas Hiessl, Vasileios Karagiannis, Christoph Hochreiner, Stefan Schulte
Distributed Systems Group, TU Wien, Austria
thomashiessl@gmx.at
{v.karagiannis, c.hochreiner, s.schulte}@infosys.tuwien.ac.at
Matteo Nardelli
Department of Civil Engineering and Computer Science Engineering
University of Rome Tor Vergata, Italy
nardelli@ing.uniroma2.it

***Abstract*—Elastic data stream processing enables applications to query and analyze streams of real time data. This is commonly facilitated by processing the flow of the data streams using a collection of stream processing operators which are placed in the cloud. However, the cloud follows a centralized approach which is prone to high latency delay. For avoiding this delay, we leverage on the fog computing paradigm which extends the cloud to the edge of the network.**

In order to design a stream processing solution for the fog, we first formulate an optimization problem for the placement of stream processing operators, which is tailored to fog computing environments. Then, we build a plugin (for stream processing frameworks) which solves the optimization problem periodically in order to support the dynamic resources of the fog. We evaluate this approach by performing experiments on an OpenStack testbed. The results show that our plugin reduces the response time and the cost by 31.5% and 8.8% respectively, compared to optimizing the placement of operators only upon initialization.

***Index Terms*—Edge Computing, Stream Processing, Internet of Things**

I. Introduction

Internet of Things (IoT) applications aim at increasing the quality of life in modern societies by improving domains such as healthcare, transportation and industry [1]. To facilitate these applications, a plethora of sensing devices produces an enormous amount of data which is usually processed in the cloud [2]. In order to handle this great deal of data in (near) real-time, cloud computing infrastructures commonly employ Data Stream Processing (DSP) frameworks like Apache Storm or Apache Spark [3]. Such frameworks are responsible for deploying and maintaining DSP *topologies* which describe the lifecycle of the data [4]. Specifically, a DSP topology consists of data sources generating data streams and operators collecting the streams and performing well-defined operations such as data filtering, processing or storing [5].

The paper at hand argues that instead of deploying DSP topologies in the cloud which might bear high latency delay [6], it may be beneficial to deploy the topologies in the fog. This is different because the fog extends the cloud by including compute resources at the edge of the network [7]. While DSP at the edge of the network has already been discussed before [6], [8], there is—to the best of our knowledge—no scheduling algorithm for the placement of operators, that has been originally designed to address fog computing environments. Thus, aspects such as the dynamic nature of the fog resources in volatile IoT environments, are not taken into account. However, there are approaches that formulate optimization problems which integrate logic to replace the scheduling algorithms of existing DSP frameworks in order to address fog computing [9], [10]. Even though the existing work provides fundamental insights regarding the enactment of DSP in the fog, the formulation of the existing objective functions does not consider leased resources or migration cost. These are important aspects of fog computing [11], [12].

Therefore, within this paper we formulate an optimization problem which considers various Quality of Service (QoS) attributes (e.g., response time, availability, enactment and migration cost) in order to address fog computing environments. To this end, we extend an existing Integer Linear Programming (ILP) problem [13] by modeling additional attributes of the fog (e.g., leased resources). Moreover, we integrate the formulation of the optimization problem within a plugin (for stream processing frameworks) which is compatible with the *Vienna Ecosystem for Elastic Stream Processing* (VISP) [14]. This plugin solves the optimization problem and provides DSP frameworks with commands to apply the optimized solution. Notably, the proposed plugin allows performing the optimization periodically during the runtime of a fog-based DSP system. In the evaluation (cf. Section V), we show that in a fog computing environment, this plugin in combination with the presented optimization model, lowers the response time and the cost, compared to optimizing statically upon initialization, i.e., during design or deployment time.

The rest of the paper is organized as follows: The next section presents existing work which is used as foundation for our approach to implement fog-based DSP. Section III discusses the system model and the actual optimization problem. Then, in Section IV, we present the internal structure of the plugin which integrates the optimization model. Afterwards,

This work was partially funded by the European project H2020 FORA (Grant Agreement: 764785).

978-1-7281-2365-3/19/$31.00 ©2019 IEEE

in Section V, we build a testbed based on OpenStack and we evaluate the performance of the proposed plugin based on a series of experiments which focus on response time and cost. Finally, Section VI presents related work and Section VII concludes the paper and proposes future research directions.

II. Background

This section presents the background of our approach. Section II-A discusses briefly the optimization problem of placing operators on distributed resources and Section II-B describes VISP.

A. Optimal Placement of Operators

Cardellini et al. [13] formulate the optimal DSP placement (ODP) which is an ILP problem for optimizing the placement of DSP operators. The objective function in ODP, considers QoS attributes such as latency and availability. Each QoS attribute is accompanied by a modifiable weight so that the optimization goal can be adjusted according to the applications' requirements. The available and the required processing resources are also taken into account in order to cope with resource heterogeneity. The developed optimization model is integrated into Apache Storm and is executed upon the initialization of a DSP application. The output of the optimization is sent to a scheduler which places the operators of an Apache Storm topology on the available processing resources.

Even though ODP targets distributed environments (e.g., fog computing), the utilized objective function does not consider leased resources or migration cost. Moreover, optimization is only performed during deployment time, i.e., dynamic changes during runtime are not taken into account. For these reasons, we extend the ODP and formulate a new optimization problem, i.e., the optimal DSP replacement (ODR) in Section III-C.

B. The VISP ecosystem

In our former work, we present VISP which is an ecosystem for elastic DSP in the IoT [14]. VISP consists of two prime components: the marketplace and the runtime. The former is a platform for hosting the executable files of operators and for designing DSP topologies. The runtime pulls the operators from the marketplace and performs actions such as instantiating, executing and monitoring the operator instances. These instances are deployed on cloud resources, e.g., using Virtual Machines (VMs). VISP provides the functionality of a full-fledged research stream processing engine that includes a *data provider* which helps the benchmarking of stream processing. For this reason, we select VISP as a base for the development of a plugin that adapts DSP to fog computing. However, it should be noted that the work at hand can be integrated into any DSP framework (e.g., Apache Storm or Apache Spark) which allows adding the necessary interactions for applying optimizations on a DSP topology.

VISP includes an elasticity component named the *reasoner*, for instantiating additional operators during runtime. However, this component does not consider resources that become available/unavailable after the deployment of the operators.

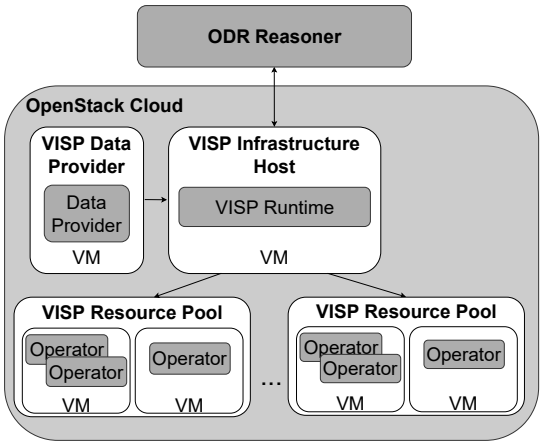

Fig. 1: A deployment of VISP using the ODR reasoner.

Therefore, to take into account such resources, a new placement method has to be designed. To this end, we introduce the *ODR reasoner* plugin for VISP, which effectively replaces the existing reasoner. This plugin integrates the formulation of ODR and solves the optimization problem periodically during runtime in order to support dynamic resources.

Fig. 1 depicts an example deployment of VISP using the ODR reasoner. In this figure, VISP remains responsible for deploying the operators while the ODR reasoner interacts only with the VISP runtime. Specifically, in Fig. 1, VISP is hosted on VMs which represent the available compute nodes in the cloud and the fog. The data provider is used for providing the data streams to the topologies which have been enacted by the VISP runtime. The runtime manages the available resources and provides an interface for the ODR reasoner. The ODR reasoner polls for monitoring information and optimizes the placement of the operators periodically. After each optimization cycle, the ODR reasoner reports back to the runtime with commands to improve the performance of the DSP topology. Such commands may include instantiating more operators or migrating an operator to another VM.

III. Problem Formulation

In this section, we extend the system model and the problem formulation of ODP [13] in order to design the ODR problem. For this, we first present the ODP system model (Section III-A) including additional parameters which help in the formulation of ODR. Then, we briefly describe the ODP problem (Section III-B). After that, we present the formulation of ODR (Section III-C). Specifically, ODR extends ODP with aspects such as: resource constraints and processing duration of the operators as well as enactment and migration cost. For better understanding of the system model and the problem formulation, Table I provides an overview of the used notation.

A. System Model

Our underlying fog system model includes fog-based compute resources and applications running on these resources. The available resources are represented by a graph G_{res} that

consists of vertices V_{res} marking the compute nodes and edges E_{res} marking the network links between the vertices [13]. Assuming two compute nodes u and v with $u, v \in V_{res}$, we define the following QoS attributes. The availability of a compute node and a network link are denoted as A_u and $A_{(u,v)}$, respectively. The delay of a network link is noted as $d_{(u,v)}$. The speedup of a compute node S_u indicates the speed of processing on u. S_u is used in order to classify the compute nodes based on resource capacity. We model three resource classes as $SMALL$, $MEDIUM$ and $LARGE$, which contain compute nodes with similar capacities and therefore, corresponding speedups sp_{SMALL}, sp_{MEDIUM} and sp_{LARGE}. Thus, if $u \in MEDIUM$ and $v \in SMALL$, the same operator runs faster on u, i.e., $S_u > S_v$.

A DSP application is represented by a topology graph G_{dsp} that consists of vertices V_{dsp} marking the operators and edges E_{dsp} marking the flow of the streams between the operators [15]. The speedup that an operator i experienced at the deployment location of the previous optimization cycle is denoted as S_i. Based on the value of S_i, we distinguish among different resource categories as discussed later in Section III-C.

We further extend this model with additional notation for the formulation of ODR. The enactment cost and the migration cost are modeled as C_u and $C_{(i,u,v)}$, respectively. Since the fog may include heterogeneous resources, we also model CPU frequency $P_{(CPU,u)}$, number of CPU cores $P_{(Cores,u)}$, memory capacity $P_{(Mem,u)}$ and storage capacity $P_{(HD,u)}$.

Each operator has the following resource requirements: CPU frequency $P_{(CPU,i)}$, number of CPU cores $P_{(Cores,i)}$, memory $P_{(Mem,i)}$ and storage $P_{(HD,i)}$. We also consider the operator image size s_i which affects the cost in case of migrations from one compute node to the other. The execution time of processing k data tuples on the operator $i \in V_{dsp}$ is modeled as ET_i. The number k results from the number of tuples that is received from all operator predecessors when exactly one data tuple is sent by the data source. Apart from ET_i, we also use $T_{(actual,i)}$ to denote the monitored value of the execution time of processing k data tuples on the operator i. It should be noted that $T_{(actual,i)}$ also includes queuing delays. This is used later in Section III-C along with the maximum limit of $T_{(actual,i)}$ that we note as $T_{(max,i)}$. These are used for detecting bottlenecks i.e., when $T_{(actual,i)} > T_{(max,i)}$.

Since the operator images are initially hosted in the VISP marketplace M, we model the transfer of the images from the marketplace to the compute nodes. We denote $b_{(M,u)}$ as the data rate of transferring the image from M to $u \in V_{res}$. The placement of an operator i on a compute node u is modeled as a decision variable $x_{i,u}$. In addition, the placement variable of the previous optimization cycle is denoted as $x_{i,u}^{prev}$. We use $x_{i,u}^{prev}$ in order to calculate the migration cost from transferring an operator i from one compute node to the other.

B. Optimal DSP Placement

According to Cardellini et al. [13] the response time of a DSP topology is defined as the maximum delay of all the paths in the flow of a data stream as described in Equation 1.

TABLE I: Notation of the problem formulation

Symbol	Description
G_{dsp}	Graph representing the DSP topology
E_{dsp}	Edges of G_{dsp} (streams)
V_{dsp}	Vertices of G_{dsp} (operators)
G_{res}	Graph representing the available resources
V_{res}	Vertices of G_{res} (compute nodes)
E_{res}	Edges of G_{res} (logical links)
ET_i	Execution time (sec) of $i \in V_{dsp}$ (per tuple)
C_u	Enactment cost of $u \in V_{res}$ per second
$C_{(i,u,v)}$	Cost for migrating $i \in V_{dsp}$ from $u \in V_{res}^i$ to $v \in V_{res}^i$
s_i	Size (MB) of image of $i \in V_{dsp}$
$T_{(actual,i)}$	Actual processing time of $i \in V_{dsp}$ (per tuple)
$T_{(max,i)}$	Maximum processing time of $i \in V_{dsp}$ (per tuple)
$P_{(Cores,u)}$	Available number of cores in $u \in V_{res}$
S_u	Processing speedup of $u \in V_{res}$
S_i	Processing speedup of $i \in V_{dsp}$ in previous optimization cycle
A_u	Availability of $u \in V_{res}$
$A_{(u,v)}$	Availability of $(u,v) \in E_{res}$
$d_{(u,v)}$	Network delay (sec) of $(u,v) \in E_{res}$
M	Node hosting the marketplace $M \notin V_{res}$
$b_{(M,u)}$	Data rate (MB/s) between M and $u \in V_{res}$
$x_{i,u}$	Placement of $i \in V_{dsp}$ on $u \in V_{res}$
$y_{(i,j)(u,v)}$	Placement if $(i,j) \in E_{dsp}$ on $(u,v) \in E_{res}$
$x_{i,u}^{prev}$	Placement of $i \in V_{dsp}$ on $u \in V_{res}$ in the previous optimization cycle
R	Response time of a topology G_{dsp}
$R_i(x)$	Response time of an operator $i \in V_{dsp}$
$V_{res}^i \subseteq V_{res}$	Subset of nodes where $i \in V_{dsp}$ can be placed

A path is a sequence of edges connecting multiple vertices. The delay of a path R_p is defined in Equation 2 whereby the first addend reflects the time spent for processing tuples within all operators along the path and the second addend reflects the delay of sending data tuples within the network. The $R_i(x)$ in Equation 3 denotes the response time of a single operator and the $D_{(i,j)}(y)$ in Equation 4 denotes the delay between operator i and j. This definition of response time applies also for the formulation of ODR (in Section III-C).

$$R = \max_{p \in \pi_{G_{dsp}}} R_p(x,y) \tag{1}$$

with

$$R_p(x,y) = \sum_{k=1}^{n_p} R_{i_k}(x) + \sum_{k=1}^{n_p-1} D_{(i_k,i_{k+1})}(y) \tag{2}$$

$$R_i(x) = \sum_{u \in V_{res}^i} \frac{ET_i}{S_u} x_{i,u} \tag{3}$$

$$D_{(i,j)}(y) = \sum_{(u,v) \in V_{res}^i \times V_{res}^j} d_{(u,v)} y_{(i,j),(u,v)} \tag{4}$$

The availability of a DSP topology assumes independent compute nodes $u \in V_{res}$ and network links $(i,j) \in E_{res}$ as expressed in Equation 5. Since quadratic or higher order equations cannot be solved as ILP problems, multiplications of the decision variables $x_{i,u}$ and $y_{(i,j),(u,v)}$ are avoided by using the logarithm of the DSP availability. Equation 8 describes the logarithm applied on Equation 5, with $\log(A_u(x)) = a_u$

and $\log(A_u(x,y)) = a_{u,v}$. Even though this application is not correct in all cases, it holds for the ODP model [13]. The same definition of availability applies also for the formulation of ODR (in Section III-C).

$$A(x,y) = \prod_{i \in V_{dsp}} A_i(x) \cdot \prod_{(i,j) \in E_{dsp}} A_{(i,j)}(y) \tag{5}$$

with

$$A_i(x) = \sum_{u \in V^i_{res}} A_u x_{i,u} \tag{6}$$

$$A_{(i,j)}(y) = \sum_{(u,v) \in V^i_{res} \times V^j_{res}} A_{(u,v)} y_{(i,j),(u,v)} \tag{7}$$

$$\log A(x,y) = \sum_{i \in V_{dsp}} \sum_{u \in V^i_{res}} a_u x_{i,u} + \\ + \sum_{(i,j) \in E_{dsp}} \sum_{(u,v) \in V^i_{res} \times V^j_{res}} a_{(u,v)} y_{(i,j),(u,v)} \tag{8}$$

1) Objective Function: The objective function of ODP is described in Equation 9 which focuses on response time and availability. This objective function is formulated according to the Simple Additive Weighting (SAW) method which simplifies the addition of QoS attributes [16]. According to SAW, objective functions use different weights for each QoS attribute. Each weight w_i has a value such that the sum $\sum_i w_i = 1$. The influence of each QoS attribute in Equation 9 is balanced using normalization. To normalize the values of the QoS attributes, a division by $R_{\max} - R_{\min}$ and $\log A_{\max} - \log A_{\min}$ is performed as shown in Equation 9. $R_{\max}$ and $R_{\min}$ are the maximum and minimum response times of a DSP topology.

Similarly, $\log A_{\max}$ and $\log A_{\min}$ represent the maximum and minimum availability. The variable r denotes $R(x,y)$ and the constraint of Equation 10 ensures that r is greater or equal to the response times of all the topology paths. Considering that r has to be minimized, it is valid to have $r = \max_{p \in \pi_{G_{dsp}}} R_p(x,y) = R(x,y)$.

$$\max_{x,y,r} F'(x,y,r) \quad \text{with}$$

$$F'(x,y,r) = w_r \frac{R_{\max} - r}{R_{\max} - R_{\min}} + \\ + w_a \frac{\log A(x,y) - \log A_{\min}}{\log A_{\max} - \log A_{\min}} \tag{9}$$

subject to:

$$r \geq \sum_{k=1}^{n_p} \sum_{u \in V^{i_k}} \frac{ET_i}{S_u} x_{i_k,u} + \\ + \sum_{k=1}^{n_p - 1} \sum_{(u,v) \in V^{i_k}_{res} \times V^{i_{k+1}}_{res}} d_{(u,v)} y_{(i_k,i_{k+1}),(u,v)} \ \forall p \in \pi_G \tag{10}$$

Other constraints which apply on Equation 9 are: *i*) The required resources of an operator i are less than the provided resources of the compute node u that hosts i. *ii*) One compute node hosts up to one operator. *iii*) If $y_{(i,j),(u,v)}$ equals to one, operators i and j are deployed on nodes u and v, respectively. *iv*) The variables $x_{i,u}$ and $y_{(i,j),(u,v)}$ which represent the placement decisions after the optimization, are Boolean.

C. Optimal DSP Replacement

In this section, we extend the formulation of ODP to better fit fog computing environments. To this end, we integrate the formulations of: migration cost ($C_{(i,u,v)}$), enactment cost (C_u) and processing time ($T_{(actual,i)}$) which have been defined in Section III-A. Specifically, we formulate ODR as follows.

The *enactment cost* is the cost of running DSP topologies on the available resources. We model $C_{op}(x)$ as total enactment cost per second as shown in Equation 11.

$$C_{op}(x) = \sum_{i \in V_{dsp}} \sum_{u \in V^i_{res}} C_u x_{i,u} \tag{11}$$

The *migration cost* is the cost that derives from all planned migrations after each optimization cycle. This cost is shown in Equation 12 as the sum of each single migration cost $C_{(i,u,v)}$. The single migration cost is described in Equation 13 and considers the operator image size s_i and the data rate $b_{(M,v)}$ of pulling the operator image from the marketplace to a compute node. The division of these two variables (s_i and $b_{(M,v)}$) equals to the duration of loading the image in the destination node $v \in V^i_{res}$. The multiplication of this duration ($v \in V^i_{res}$) with C_u represents the cost of migrating an operator i to a node u.

$$C_{mig}(x) = \sum_{i \in V_{dsp}} \sum_{u \in V^i_{res}} \sum_{v \in V^i_{res} \setminus \{u\}} C_{(i,u,v)} x^{prev}_{i,u} x_{i,v} \tag{12}$$

$$C_{(i,u,v)} = \frac{s_i}{b_{(M,v)}} C_u \tag{13}$$

For response time and availability, we use the definitions of ODP as shown in Equations 1-4 and 5-8, respectively. However, we modify the speedup used in Equation 3 to support various resource classes. Thus, the speedup S_u is assigned with one of the three speedup values according to the resource class of the node $u \in V_{res}$.

$$S_u = \begin{cases} sp_{small} & \text{if } u \in SMALL \\ sp_{medium} & \text{if } u \in MEDIUM \\ sp_{large} & \text{if } u \in LARGE \end{cases} \tag{14}$$

1) Objective Function: Following the example of ODP which uses the SAW technique, we add three more weighted QoS attributes to the objective function of Equation 9. These are: the *budget* constraints, the *processing duration* constraint and the *resource* constraints, as explained below. The resulting objective function of ODR is shown in Equation 15.

$$F'_{cost} = F'(x,y,r) + w_{c_{op}} \frac{C_{op\max} - C_{op}(x)}{C_{op\max} - C_{op\min}} + \\ + w_{c_{mig}} \frac{C_{mig\max} - C_{mig}(x)}{C_{mig\max} - C_{mig\min}} \tag{15}$$

The *budget constraints* are used for limiting the amount of the enactment cost. We use the variable B in Equations 16

and 17 as a limit for the enactment cost B_{op} and the migration cost B_{mig}, respectively. This is done to penalize the operator replacements in order to regulate the cost and keep the performance at a certain level [17].

$$C_{op}(x) \leq B_{op} \tag{16}$$

$$C_{mig}(x) \leq B_{mig} \tag{17}$$

The *processing duration constraint* serves as a trigger for redeploying operators. This constraint ensures that operators are deployed on compute nodes with enough resources (i.e., higher speed-up S_u) and that the processing duration does not exceed a maximum limit. We use this constraint to avoid bottlenecks when the preceding operators send many tuples to an overloaded operator. In this case, migrating the overloaded operator to a node of a higher category (e.g., medium or large) which processes tuples faster, reduces the processing duration.

To formulate the maximum limit of the processing duration, we consider the processing duration of tuples as shown in Equation 18. The left-hand side of the equation is the processing duration that results if operator i is deployed on a compute node u with speedup S_u. This side has to be less or equal to the limit $T_{max,i}$, which resides in the right-hand side of the equation.

$$\sum_{u \in V_{res}} T_{(actual,i)} \frac{S_i}{S_u} x_{i,u} \leq T_{(max,i)} \quad \forall i \in V_{dsp} \tag{18}$$

The *resource constraints* replace the generic resource capacity constraint of ODP [13]. Specifically, Equations 19-21 model CPU, memory and storage, respectively. Equation 19 ensures that the required CPU of the operator is less or equal to the CPU capacity of the host compute node. CPU is considered as the product of the number of CPU cores and the CPU frequency (this product is also referred to as *vCPU*). CPU is formulated this way so that operators with high requirements of CPU frequency can also be hosted in nodes that have low CPU frequency but are equipped with many CPU cores. Similarly, high CPU frequency can cover the shortage of CPU cores. Finally, Equations 20 and 21 ensure that the resource requirements of an operator are less or equal to the capacity of the host compute node for memory and storage, respectively.

$$\sum_{i \in V_{dsp}} P_{(CPU,i)} P_{(Cores,i)} x_{i,u} \leq P_{(CPU,u)} P_{(Cores,u)} \quad \forall u \in V_{res} \tag{19}$$

$$\sum_{i \in V_{dsp}} P_{(Mem,i)} x_{i,u} \leq P_{(Mem,u)} \quad \forall u \in V_{res} \tag{20}$$

$$\sum_{i \in V_{dsp}} P_{(HD,i)} x_{i,u} \leq P_{(HD,u)} \quad \forall u \in V_{res} \tag{21}$$

IV. The ODR Reasoner

In this section, we present the ODR reasoner which integrates the optimization problem of ODR (as presented in Section III). The internal structure of the ODR reasoner comprises the following components (cf. Figure 2):

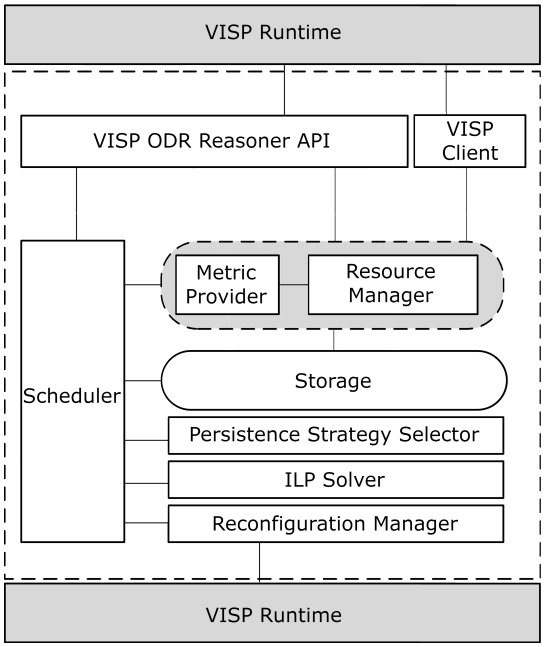

Fig. 2: The internal structure of the ODR reasoner.

The *VISP ODR Reasoner API* is the northbound interface for receiving optimization requests from the VISP runtime. As an optimization request, we consider the initial call that VISP used to make to the predecessor of the ODR reasoner, i.e., the *reasoner* (cf. Section II), which is now redirected to the ODR reasoner. After the submission of an optimization request to the API, a *Metric Provider* and a *Resource Manager* are instantiated. The Resource Manager is responsible for the parameters of the system model, which are related to the available resources. The Metric Provider is responsible for the following variables:

- Response time boundaries for $R_{\max}$ and $R_{\min}$ assigned by monitoring the response time R (cf. Equation 1). These values are used for the normalization of the QoS attributes in the objective function (cf. Equations 9 and 15).
- Availability boundaries for $A_{\max}$ and $A_{\min}$ assigned by monitoring the availability $A(x, y)$ (cf. Equation 5). These values are used for the normalization of the QoS attributes in the objective function (cf. Equations 9 and 15).
- Enactment cost boundaries for $C_{op\,\max}$ and $C_{op\,\min}$ representing the max and min enactment cost $C_{op}(x)$ (cf. Equation 11). These are used for the normalization of the QoS attributes in the objective function (cf. Equation 15).
- Migration cost boundaries for $C_{mig\,\max}$ and $C_{mig\,\min}$ which represent the max and min migration cost $C_{mig}(x)$ calculated according to Equation 12. These values are used for the normalization of the QoS attributes in the objective function (cf. Equation 15).
- Execution time ET_i for each operator $i \in V_{dsp}$. These values are assigned by monitoring the deployment of the DSP topology on the available resources.
- Network delay $d_{(u,v)}$ for all links $(u, v) \in E_{res}$. These

values are requested from the VISP runtime.
- Speedup S_u for each $u \in V_{res}$. These are assigned based on the size of the node u (small, medium or large). The resource capacities are requested from the VISP runtime.

Both the Resource Manager and the Metric Provider request the required data from the *VISP Client* which pulls data from the VISP runtime. All data is stored in the form of graphs, metrics, and optimization settings to an in-memory *Storage*.

After storing the data, the *Scheduler* which is triggered periodically (based on fixed time intervals), passes the optimization parameters to the *ILP Model Solver*. This component integrates an ILP solver which returns the optimized placement according to the formulation of ODR. To ensure that all the dynamic changes are incorporated into the optimization model, the ILP Model Solver does not consider information from previous optimization cycles. After solving the optimization problem, the optimized placement is converted to a VISP-compatible format by the *Reconfiguration Manager* which sends the placement commands to the VISP runtime.

Notably, before passing the optimization parameters to the ILP Model Solver, the Scheduler invokes the *Persistence Strategy Selector*. This component limits the amount of operator replacements by either pinning an operator to a node or by minimizing the migration cost. This is done for achieving only partial deployment persistence of the optimized placement over time. Deployment persistence refers to limiting the operator replacements because frequent changes are costly and may lead to inefficient resource utilization [17].

To make the ODR reasoner suitable for environments that process huge amounts of data (e.g., in IoT and fog computing scenarios), we enable the handling of many DSP topologies at the same time. To achieve this, each optimization request is assigned to an ID. Every time the Scheduler invokes the aforementioned components, the ID of the corresponding optimization request is used as a parameter. This way, multiple topologies can be saved in the Storage and be optimized periodically. This feature also enables one instance of the ODR reasoner to be able to handle many VISP runtimes.

V. Evaluation

This section provides the evaluation of the ODR reasoner. First, Section V-A describes the fog computing environment we set up. Then, Section V-B presents results from running the ODR reasoner in this environment. The results we show focus on *response time* (as defined in Equation 1), *total cost* (including enactment and migration cost as defined in Equations 11 and 12, respectively) and *total score* which represents the maximized value of the objective function (as defined in Equation 15). These specific metrics are selected in order to show that the ODR reasoner lowers the response time and the cost, compared to optimizing statically only upon initialization.

A. Evaluation Environment

To evaluate the ODR reasoner, we set up an OpenStack-based private cloud. OpenStack is used for creating VMs to host the VISP ecosystem and the operators. Three types of VMs are provisioned with different resource capacities: *i*) m1.medium with vCPU = 3 (product of CPU cores and CPU frequency in GHz), memory = 2 GB and storage = 40 GB. *ii*) m2.medium with vCPU = 5, memory = 3 GB and storage = 40 GB. *iii*) m1.xlarge with vCPU = 15, memory = 5 GB and storage = 40 GB. The ODR reasoner is developed as a VISP plugin in Java, using the Spring Boot library for implementing the API that enables communication with VISP.

For solving the proposed optimization problem, the ODR reasoner requires an ILP solver (as explained in Section IV) such as the IBM CPLEX, the GUROBI or the MOSEK. In our case, we use the Java API of the IBM CPLEX due to its compatibility with the ODR reasoner (which is also developed in Java) and the intuitive interfaces. Using CPLEX we can create an ILP optimization model, define the objective function and add various constraints. Specifically, the ILP solver first instantiates *IloLinearNumExpr* objects which are used for expressions of numerical variables of any type (e.g., resource capacities). Then, to build an optimization model using these variables, an IloModeler object is used. The IloModeler is also used for constructing the constraints and the objective function. Finally, an *IloCplex* object which integrates the tools to solve various mathematical programming models, collects all the modeled expressions in order to be able to solve the optimization problem of ODR.

To ensure that the ILP solver produces a satisfactory solution within a reasonable amount of time despite the dynamic changes of the fog resources, we configure CPLEX to stop optimizing when at least one out of two conditions is met: *i*) The processing duration exceeds a predetermined time limit. *ii*) The value of the objective function reaches a predetermined threshold. In both cases, CPLEX stops and returns the maximized solution.

To emulate a fog computing environment, we set up three VISP runtimes (using Redis, MySQL and RabbitMQ for achieving reliable communication/storage) as shown in Fig. 3. One runtime is considered to be in the cloud managing two VMs ($u \in V_{res}$). The other two runtimes are considered to be fog nodes and each one manages one VM ($u \in V_{res}$). To emulate the dynamic resources of a fog environment, one fog node joins the network 20 minutes after the beginning of the experiments. This is necessary for the system to stabilize so that the observed changes after this time period, can be attributed to the dynamic resources.

Moreover, we manipulate the network delays among the compute nodes using the Traffic Control (tc) utility[1]. The communication between the cloud VMs bears no network delay whereas, reaching a fog VM from the cloud requires 400 ms of delay, due to the remote location. The fog VMs are assumed to reside close to each other and thus, the communication between them requires 10 ms of delay.

For this fog computing environment, we assume a DSP topology with four operators, a data source, and a data sink

[1]https://linux.die.net/man/8/tc

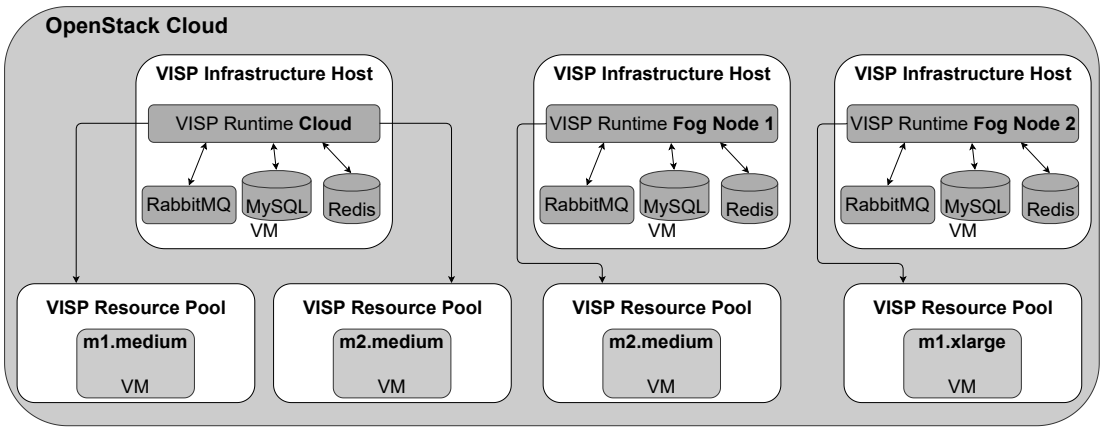

Fig. 3: The fog computing environment used in the evaluation.

as shown in Fig. 4. The source is configured to generate data tuples with sinusoidal frequency (on average, one message per second) to account for different load cases. The operators are configured to have fixed processing times ET_i of 100, 250, 500 and 1000 ms for operator 1, 2, 3 and 4, respectively. Each operator starts with size: *small*, because scaling to *medium* and *large* occurs during runtime (if needed) by the ODR reasoner.

The ODR reasoner repeats the optimization every 4 minutes which allows time for the system to stabilize after each optimization cycle. For this evaluation, we allow the system to operate for 50 minutes consecutively, so that the effect of the optimizations can be noticed through the observed metrics. Moreover, the results we present in Section V-B are based on the average values from running 3 experiments (each one lasting 50 minutes).

To take into consideration the different aspects of the cost related to placement, we account for depreciation, maintenance and leasing, which all add up to the total cost that is measured in currency units (CU) per second. *Depreciation* refers to the allocated cost of an asset (e.g., hardware resources) based on the initial purchase expense and the duration over which the investment is amortized [18]. *Maintenance* refers to the cost of keeping the resources at a functional state without defects.

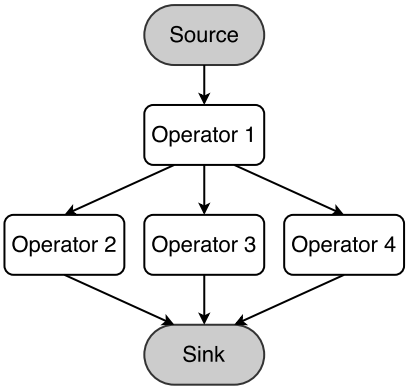

Fig. 4: The DSP topology used in the evaluation.

Finally, the *Leasing* cost is a price that has to be paid to the owner of the resources (if the resources are leased).

The cloud VMs can be leased for a cost of 15.5 CU/s for *m1.medium* and 20.5 CU/s for *m2.medium*. These numbers represent the cost of using cloud services whereby the cost of a VM is associated with the respective compute resources. Since these resources are not owned, we do not consider depreciation and maintenance cost.

Fog node 1 resides on-premise (leasing: 0 CU/s) and has a maintenance cost of 29.5 CU/s. This cost is higher than a VM of the same compute resources in the cloud because dedicated on-premise resources are more expensive than cloud resources [19]. Fog Node 2 also resides on-premise, but the associated compute resources are shared among multiple clients. Since the execution of data processing in fog node VMs is not free of charge [20], the maintenance cost is shared among all the clients and thus it remains low (1.5 CU/s). Both fog nodes have depreciation cost of 1 CU/s. We also consider an optimization server which hosts the ODR reasoner and solves the optimization problem for a total *optimization cost* of 3 CU/s.

To provide a baseline for the evaluation, apart from executing the ODR reasoner (dynamic optimization), we also perform the same experiments using static optimization. Static optimization executes the actions related to the optimization and placement of a DSP topology only on system startup. Since the motivation for periodic optimization is due to the dynamic resources of the fog, having a baseline that represents static optimization may reveal if the periodic approach is actually more suitable for fog computing. A more detailed technical analysis of the evaluation environment as well as the implementation of the ODR reasoner can be found in [21].

B. Evaluation Results

In this section, we present the average values (standard deviations are shown in Table II) of the evaluation results which focus on response time (Section V-B1), total cost and total score (Section V-B2). Moreover, we show a summary of the values of all the QoS attributes in Section V-B3.

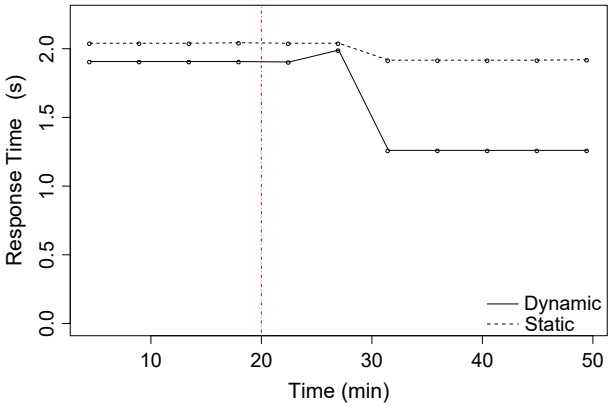

Fig. 5: Response time when the objective function considers equal QoS attributes.

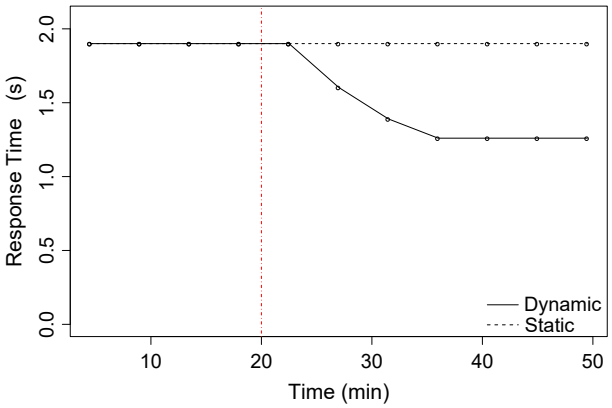

Fig. 6: Response time when the objective function focuses on minimizing response time.

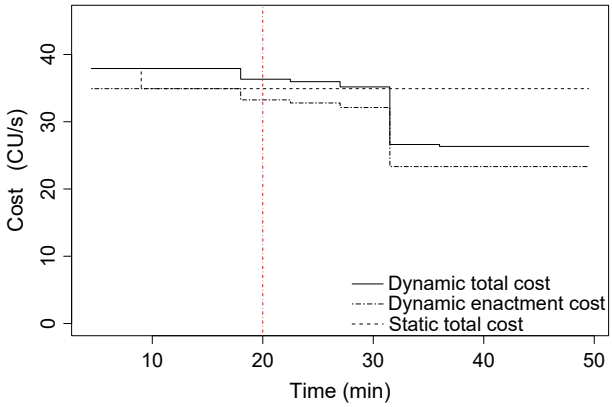

Fig. 7: Cost when the objective function considers equal QoS attributes.

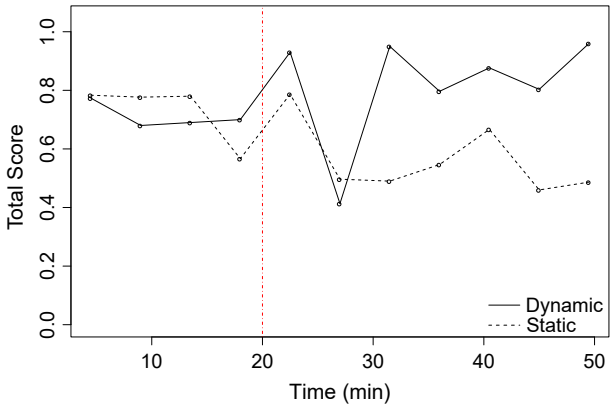

Fig. 8: Total score when the objective function considers equal QoS attributes.

1) Response Time: Fig. 5 shows the response time of the DSP topology (as defined in Equation 1) throughout the 50 minutes of the evaluation. We notice that the response time of the dynamic optimization (ODR) is permanently lower than the response time of the static optimization. What is interesting in this figure, is that the dynamic line starts lower. The explanation for this is that the monitoring data on startup are not exactly the same for the two cases. This can lead to changes in the boundary parameters (e.g., R_{max} and R_{min}, cf. Section IV), which in turn influence the outcome of the objective function (cf. Equation 15). For this reason, the static optimization achieves higher total score when focusing on other QoS attributes of the objective function (e.g., cost, availability). This can lead to placements that are more cost-efficient but bear longer response time.

When fog node 2 becomes available (after 20 minutes, as indicated by the horizontal red line in Fig. 5), the ODR reasoner performs replacements which significantly decrease the response time to 1.3 seconds. Compared to the 1.9 seconds of the static optimization, the reduction achieved by ODR amounts to 31.5%. This reduction derives from the ODR reasoner because it exploits the fog node 2 during runtime by moving operators to the fog. This lowers the response time of the DSP topology since the communication between the fog nodes has low delay.

To acquire the results of Fig. 5, we configure all the QoS attributes of the objective function (cf. Equation 15) to have equal weights (i.e., $w_r = w_a = w_{c_{op}} = w_{c_{mig}} = 0.25$). By modifying these weights, we can configure ODR to achieve optimality by focusing on different QoS attributes.

To unveil the maximum potential of ODR with regard to the response time, we run the same experiment again, but modify the weights of the objective function to focus on response time (i.e., $w_r = 1$ and $w_a = w_{c_{op}} = w_{c_{mig}} = 0$). We plot the results in Fig. 6 which shows that contrarily to Fig. 5, both lines start with the same value of response time (at 1.9 seconds). This happens because in this case, both approaches achieve optimality by lowering the response time.

When the fog node 2 becomes available in Fig. 5, we notice that the line drops for three optimization cycles in a row until it reaches a response time of 1.26 seconds (i.e., 33.5% reduction). Notably, the response time drops for three optimization cycles even though ODR aims at achieving global optimum in every cycle. This happens because the monitoring data reported to ODR after executing the initial optimization, aid in applying further improvements.

2) Total Cost and Total Score: Fig. 7 shows the value of the cost (CU/s) throughout the time of the experiments. The dashed line represents the total cost of the static optimization which consists of the enactment cost and the initial optimization cost. The dot-dash line represents the enactment cost of ODR and the solid line represents the total cost of ODR which is the sum of the enactment, migration and optimization cost. To acquire the results of Fig. 7, we configure equal weights for the QoS attributes.

In Fig. 7, we notice that the enactment and total cost of ODR exhibit similar behavior. This happens because the migration cost is low compared to the enactment cost. Thus, the difference in these two lines is due to the fixed optimization cost which is charged per second. Regarding the static optimization, the optimization cost decreases to 0 after the first optimization cycle because the respective resource is released.

By comparing dynamic total cost with static total cost, we see that the ODR bears the additional optimization cost. However, this cost is compensated by the placement decisions which take place after the available resources change (i.e., fog node 2 becomes available). Notably, even before fog node 2 becomes available, ODR finds a placement that costs less than the initially optimized placement. This is visible through the line that shows the ODR enactment cost which is lower than the static line right before fog node 2 appears. Hence, ODR can reduce the cost not only due to changes in the available resources but also, by integrating the monitoring data in the optimization process. The total cost savings of ODR compared to the static optimization amount to 3.09 CU/s (i.e., 8.8%).

Fig. 8 shows the total score based on the optimization goal (maximized value) of ODR as defined in Equation 15. Even though ODR starts at the same point with the static optimization, in the next two points it scores lower. This can be explained by the cost of the optimization which occurs periodically. In the next point, it scores higher because the monitoring data help to find changes that improve performance. After fog node 2 becomes available (vertical red line in Fig. 8), ODR scores lower due to the cost of performing the replacements that derive from the optimization due to the new available resources. From that point on, ODR scores higher for the rest of the execution time since the new resources have been taken into account. To acquire the results of Fig. 8, we configure equal weights for the QoS attributes.

3) Summary of QoS Attributes and Cost: Table II provides a summary of the results from the aforementioned experiments, including average values and respective standard deviations of all the QoS attributes. Based on this table, we notice that on average, ODR performs better than the baseline with regard to response time, availability, enactment cost, total cost and total score. For these values we use again, equal weights for the QoS attributes (i.e., $w_r = w_a = w_{c_{op}} = w_{c_{mig}} = 0.25$).

TABLE II: Average values of QoS attributes and cost.

	Static	Dynamic
Average Response Time (sec)	1.98 ($\sigma = 0.12$)	1.62 ($\sigma = 0.01$)
Average Availability	0.35 ($\sigma = 0.07$)	0.40 ($\sigma = 0.06$)
Enactment Cost (CU/s)	34.91 ($\sigma = 2.09$)	29.05 ($\sigma = 0.65$)
Migration Cost (CU/s)	-	14.47 ($\sigma = 1.17$)
Optimization Cost (CU/s)	0.27 ($\sigma = 0$)	3 ($\sigma = 0$)
Total Cost (CU/s)	35.19 ($\sigma = 2.09$)	32.10 ($\sigma = 0.65$)
Cost Savings (CU/s)	-	3.09 ($\sigma = 2.65$)
Average Total Score	0.62 ($\sigma = 0.05$)	0.78 ($\sigma = 0.04$)

VI. Related Work

As mentioned in Section I, the number of approaches aiming at applying DSP at the edge of the network is still quite low. Nevertheless, in this section we describe related works.

Sajjad et al. [6] discuss benefits from applying DSP in a decentralized manner. In this work, the authors design SpanEdge which unifies stream processing across geo-distributed resources. SpanEdge considers resources from two layers: central data centers and near-the-edge data centers. In this setting, SpanEdge places the operators near the edge with the goal to reduce latency and cost. Even though the goal of our work is similar, SpanEdge focuses on mechanisms to group the operators together so that the communication bears low latency. On the contrary, the focus of our work is to achieve low latency using formal methods, i.e., by solving an optimization problem.

Renart et al. [8] implement an edge-based DSP framework. This framework allows users to define how the data streams are processed according to origin location and content. Therefore, this work aims at applying data-driven DSP by enabling the data to form dynamic stream processing topologies at the edge. To achieve this, the authors propose an overlay network which coordinates the execution of the streaming workflows across the geographically distributed edge resources. Thus, this work focuses on a overlay network for forming the DSP topologies whereas, in our work we deploy the topologies according to the solution of an optimization problem.

Cardellini et al. [9] propose an extension to ODP [13] which we have also used as a basis in the work at hand. The resulting Optimal DSP Replication and Placement (ODRP) problem copes with the increased volume of data by replicating and placing operators on distributed compute resources. ODPR focuses on two aspects of the problem: the optimal number of replicas for each operator and the placement of each replica. Even though using replication may have the potential to minimize the response time, the cost increases with each new replica. The formulation of ODPR considers requirements such as response time, inter-node traffic, cost and availability. In our work, we also take into account the cost of enacting DSP topologies and the cost of migrating operators, which are necessary for DSP that targets fog computing environments.

Amarasinghe et al. [10] propose an optimization framework for minimizing the end-to-end latency of DSP topologies. The formulation of the optimization problem is based on R-storm [22]. The authors introduce two types of constraints, namely the *residency* constraints which are related to the location of the operator and the *resource* constraints which are related to resource utilization. On top of such constraints, in our work we also model the *processing duration* which ensures that the operators have enough resources during runtime.

Moreover, we model the *budget* constraint for limiting the cost of running DSP topologies. These extra constraints are crucial to fog computing for the case that the operators are deployed on leased resources.

Finally, Dautov et al. [23] also design a solution for stream processing at the edge. In this approach, the authors present a stream processing architecture for spreading workloads among a cluster of edge devices. The proposed solution is implemented based on Apache NiFi and is shown to perform faster than stream processing in the cloud. Dautov et al. focus on architecture and implementation details whereas in our work, apart from describing the architecture of the proposed components, we also focus on the formulation and integration of a formal optimization problem.

VII. Conclusion and Future Work

Stream processing frameworks are a popular solution for processing data in the cloud. Usually, such frameworks deploy a set of operators on cloud resources and ensure that the data is processed according to the predefined operations. In this paper, we leverage on the fog computing paradigm to perform stream processing in the fog which extends the cloud to the edge of the network. To this end, we formulate an optimization problem (ODR) that targets fog computing environments. In addition, we build a plugin (ODR reasoner) for DSP frameworks. This plugin (which is loosely coupled with VISP) performs periodic optimizations and provides frameworks with commands to exploit the fog resources. The conclusions from the evaluation of the ODR reasoner show that the cost of using fog resources can be compensated by the gain in other QoS attributes. Moreover, we notice that performing dynamic optimizations can have a positive impact on performance even when the available resources are stable. This happens because the information from monitoring the operators during runtime, can aid the optimization to find further improvements.

The goal of the evaluation in the paper at hand, is to show the general applicability of our proposed optimization approach. In the future, we plan to evaluate optimization approaches in more complex settings, e.g., with a larger number of resources and operators, and with a more volatile behavior. Regarding the optimization, the ODR is currently based on an ILP problem. However, ILPs bear concerns related to scalability and thus, studying ways to reduce the complexity of this problem is a promising research direction. Hence, we will work on heuristics that approximate the optimal solution.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A survey on application layer protocols for the internet of things," *Transaction on IoT and Cloud Computing*, vol. 3, no. 1, pp. 11–17, 2015.

[3] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, *et al.*, "Benchmarking streaming computation engines: Storm, flink and spark streaming," in *2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, pp. 1789–1792, IEEE, 2016.

[4] C. Hochreiner, M. Vögler, S. Schulte, and S. Dustdar, "Stream Processing for the Internet of Things," in *9th International Conference on Cloud Computing*, pp. 100–107, IEEE, 2016.

[5] U. Çetintemel, D. Abadi, Y. Ahmad, H. Balakrishnan, M. Balazinska, M. Cherniack, J.-H. Hwang, S. Madden, A. Maskey, A. Rasin, *et al.*, "The Aurora and Borealis Stream Processing Engines," in *Data Stream Management* (M. N. Garofalakis, J. Gehrke, and R. Rastogi, eds.), Data-Centric Systems and Applications, pp. 337–359, Springer, 2016.

[6] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, and V. Vlassov, "SpanEdge: Towards unifying stream processing over central and near-the-edge data centers," in *IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 168–178, 2016.

[7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and its Role in the Internet of Things," in *MCC Workshop on Mobile Cloud Computing*, pp. 13–16, 2012.

[8] E. G. Renart, J. Diaz-Montes, and M. Parashar, "Data-driven stream processing at the edge," in *2017 IEEE 1st International Conference on Fog and Edge Computing*, pp. 31–40, 2017.

[9] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Optimal operator replication and placement for distributed stream processing systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, no. 4, pp. 11–22, 2017.

[10] G. Amarasinghe, M. D. de Assunçao, A. Harwood, and S. Karunasekera, "A Data Stream Processing Optimisation Framework for Edge Computing Applications," in *IEEE 21st International Symposium on Real-Time Distributed Computing*, pp. 91–98, 2018.

[11] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[12] O. Osanaiye, S. Chen, Z. Yan, R. Lu, K.-K. R. Choo, and M. Dlodlo, "From cloud to fog computing: A review and a conceptual live vm migration framework," *IEEE Access*, vol. 5, pp. 8284–8300, 2017.

[13] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Optimal operator placement for distributed stream processing applications," in *10th ACM International Conference on Distributed and Event-based Systems*, pp. 69–80, 2016.

[14] C. Hochreiner, M. Vögler, P. Waibel, and S. Dustdar, "VISP: An Ecosystem for Elastic Data Stream Processing for the Internet of Things," in *20th International Enterprise Distributed Object Computing Conference*, pp. 19–29, 2016.

[15] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo, "SPADE: The System S Declarative Stream Processing Engine," in *2008 ACM SIGMOD International Conference on Management of Data*, pp. 1123–1134, 2008.

[16] K. P. Yoon and C.-L. Hwang, *Multiple Attribute Decision Making: An Introduction*, vol. 104. Sage publications, 1995.

[17] M. Woodside, Z. Li, J. Chinneck, and M. Litoiu, "Adaptive Cloud Deployment using Persistence Strategies and Application Awareness," *IEEE Transactions on Cloud Computing*, vol. 5, pp. 277–290, 2015.

[18] X. Li, Y. Li, T. Liu, J. Qiu, and F. Wang, "The method and tool of cost analysis for cloud computing," in *Cloud Computing, 2009. CLOUD'09. IEEE International Conference on*, pp. 93–100, IEEE, 2009.

[19] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 1–14, ACM, 2010.

[20] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of everything*, pp. 103–130, Springer, 2018.

[21] T. Hiessl, "Optimizing the placement of stream processing operators in the fog," *Master Thesis, TU Wien*, 2017.

[22] B. Peng, M. Hosseini, Z. Hong, R. Farivar, and R. Campbell, "R-Storm: Resource-aware Scheduling in Storm," in *16th Annual Middleware Conference*, pp. 149–161, 2015.

[23] R. Dautov, S. Distefano, D. Bruneo, F. Longo, G. Merlino, and A. Puliafito, "Pushing Intelligence to the Edge with a Stream Processing Architecture," in *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 792–799, 2017.