Joint Operator Replication and Placement Optimization for Distributed Streaming Applications

Valeria Cardellini
cardellini@ing.uniroma2.it

Vincenzo Grassi
vincenzo.grassi@uniroma2.it

Francesco Lo Presti
lopresti@info.uniroma2.it

Matteo Nardelli
nardelli@ing.uniroma2.it

Department of Civil Engineering and Computer Science Engineering
University of Rome Tor Vergata, Italy

ABSTRACT

In the last few years, several processing approaches have emerged to deal with Big Data. Exploiting on-the-fly computation, Data Stream Processing (DSP) applications can process unbounded streams of data to extract valuable information in a near real-time fashion. To keep up with the high volume of daily produced data, the operators that compose a DSP application can be replicated and placed on multiple, possibly distributed, computing nodes, so to process the incoming data flow in parallel.

In this paper, we present Optimal DSP Replication and Placement (ODRP), a unified general formulation of the operator replication and placement problem that takes into account the heterogeneity of application requirements and infrastructural resources. A key feature of ODRP is the joint optimization of the operators replication and their placement. We evaluate the proposed model through a set of numerical experiments that demonstrates its flexibility and the benefits that derive from the joint optimization.

CCS Concepts

•Software and its engineering → Distributed systems organizing principles;

Keywords

Data stream processing, Operator placement, Replication, QoS

1. INTRODUCTION

Today we have access to a huge amount of data from which we would like to extract valuable information, e.g., analysis of customers sentiment from social network data, prediction of health problems from wearable devices, optimization of the transportation system in response to social events. The volume and velocity of daily produced data have fostered the development of new processing approaches that rely on the usage of multiple computing machines. Nowadays, two opposite approaches are commonly adopted: batch processing and stream processing. The former stores all the data, usually on a distributed file system, and then operates on them on the basis of different programming models, among which the well-known MapReduce [4]. The latter processes all the data on-the-fly, i.e., without storing them, so it can produce results in a near real-time fashion. Therefore, Data Stream Processing (DSP) applications are widely used to process unbounded streams of data and timely extract valuable information. We focus on this kind of applications.

DSP applications can be represented relying on a directed acyclic graph (DAG), with data sources, operators, and final consumers as vertices, and streams as edges. Each *operator* can be seen as a black-box processing element that continuously receives incoming streams, applies a transformation, and generates new outgoing streams. In the Big Data era, DSP applications should deal with a great amount of incoming load, so there is the need of scaling their execution on multiple computing nodes, because a single machine cannot provide enough processing power. To this end, these applications usually exploit *data parallelism*, which consists in increasing or decreasing the number of parallel instances for the operators, so that each instance can process a subset of the incoming data flow in parallel (e.g., [7, 10]). Moreover, since data sources can be geographically distributed, the execution of DSP applications can also take advantage of the ever increasing presence of distributed Cloud and Fog computing resources,which can improve the system scalability and reduce latency by moving the computation towards the network edge, closer to data sources. Nevertheless, the use of such distributed infrastructure poses new challenges, that include network and system heterogeneity, geographic distribution as well as non-negligible network latencies [14].

In this paper, we address the problem of determining the degree of replication of the DSP operators and their placement over a set of distributed computing nodes. Building on our previous work [2], we present Optimal DSP Replication and Placement (**ODRP**). Differently from most works in literature [5, 11, 12, 15], **ODRP** provides a unified formulation of the replication and placement problem, which allows us to jointly optimize the placement and replication of the DSP application's operators, while maximizing the Quality of Service (QoS) attributes of the application.

VALUETOOLS 2016, October 25-28, Taormina, Italy

DOI 10.4108/eai.25-10-2016.2266628

Beside computing the optimal placement and degree of parallelism of the operators, ODRP provides a benchmark against which other centralized and decentralized placement and replication algorithms can be evaluated.

The rest of this paper is organized as follows. We review related work in Section 2; in Section 3 we describe the system model and the problem under investigation and formulate ODRP as an Integer Linear Programming (ILP) problem in Section 4. We evaluate the flexibility and the benefits of ODRP through numerical experiments in Section 5 and conclude with Section 6.

2. RELATED WORK

Most works in literature consider DSP operator placement and operator replication as independent and orthogonal decisions, where the operator placement is first carried out without determining the optimal number of replicas for each operator. Then, in response to some performance deterioration, the operators to be replicated and their new replication degree are identified. This two-stage approach requires to reschedule the DSP application in order to take the new application configuration into account and may incur in a significant overhead. In this paper, we propose a *single-stage* approach to determine both the placement and the parallelism degree of the operators in a DSP application.

The DSP placement problem has been widely investigated in literature under different modeling assumptions and optimization goals, e.g., [2, 5, 15]. In [2] we proposed a general formulation of the optimal DSP placement which takes into account the heterogeneity of computing and networking resources and which encompasses the different solutions proposed in the literature. In this paper, we extend that formulation in such a way to determine the optimal number of replicas for each operator contextually to their placement on the underlying computing and networking resources.

Since the placement problem is NP-hard, several heuristics have been proposed, e.g. [1, 13, 16]. They aim at minimizing a diversity of utility functions, such as the DSP application end-to-end latency, the inter-node traffic, and the network usage. Our problem formulation can be adjusted to take into account these different utility functions.

A consistent number of research works have focused on scaling the amount of operator replicas in response to changes observed in some monitored performance metric. Some works, e.g., [3, 9], exploit threshold-based policies based on the utilization of either the system nodes or the operator instances. Other works, e.g., [7, 11, 12], use more complex policies to determine the scaling decisions. Lohrmann et al. [11] propose a strategy that enforces latency constraints by relying on a predictive latency model based on queueing theory. Mencagli [12] presents a game-theoretic approach where the control logic is distributed on each operator.

An important issue related to operator replication regards the stateful operators, that require state migration to preserve the application integrity [7]. The approach we propose in this paper jointly places and replicates operators, thus saving the overhead and latency penalty incurred by stateful operator migrations in case of disjointed replication and placement decisions.

The works most closely related to ours have been presented by Heinze et al. [8, 10]. They propose a model to estimate the latency spike created by a set of operator movements and use it to define an operator placement algorithm based on a bin packing heuristic that minimizes the latency violations and focuses only on the placement of the newly added operators. We present an optimal problem formulation that targets the initial placement decision and can be used to benchmark existing heuristics.

So far, we have considered the replication of operators, that is a kind of scale out/in at the DSP application level and thus changes the processing graph topology; a different kind of replication exploited in literature occurs at the level of the computing resources assigned to each DSP operator [6].

Storm, Spark Streaming, and Flink, which are the most popular open-source DSP frameworks, use directed graphs to model DSP applications and therefore our problem formulation can be integrated into their scheduler; however, they require to manually tune the amount of replicas for each operator. We postpone to future work the development into Storm of the proposed approach, for whom we will exploit our previous prototypes presented in [2, 3].

Table 1: Main notation adopted in the paper.

Symbol	Description
G_{dsp}	Graph representing the DSP application
V_{dsp}	Set of vertices (operators) of G_{dsp}
E_{dsp}	Set of edges (streams) of G_{dsp}
C_i	Cost of deploying operator $i \in V_{dsp}$
R_i	Latency of $i \in V_{dsp}$ on a reference processor
Res_i	Resources required to execute $i \in V_{dsp}$
$\lambda_{(i,j)}$	Average tuple rate exchanged on $(i,j) \in E_{dsp}$
$b_{(i,j)}$	Avg. number of byte per tuple on $(i,j) \in E_{dsp}$
G_{res}	Graph representing computing and network resources
V_{res}	Set of vertices (computing nodes) of G_{res}
E_{res}	Set of edges (logical links) of G_{res}
A_u	Availability of node $u \in V_{res}$
Res_u	Amount of resources available on $u \in V_{res}$
S_u	Processing speed-up of $u \in V_{res}$
$A_{(u,v)}$	Availability of $(u,v) \in E_{res}$
$C_{(u,v)}$	Transmission cost per data on $(u,v) \in E_{res}$
$d_{(u,v)}$	Network delay on $(u,v) \in E_{res}$
$V^i_{res} \subseteq V_{res}$	Subset of nodes where $i \in V_{dsp}$ can be placed
$\mathcal{X} \sqsubset X$	Multiset of elements in X
$x_{i,\mathcal{U}}$	Placement of $i \in V_{dsp}$ on nodes in $\mathcal{U} \sqsubset V^i_{res}$
$y_{(i,j),(\mathcal{U},\mathcal{V})}$	Placement of $(i,j) \in E_{dsp}$ on the network paths from nodes in $\mathcal{U} \sqsubset V^i_{res}$ to nodes in $\mathcal{V} \sqsubset V^j_{res}$
z_u	Activation variable for $u \in V_{res}$
$z_{(u,v)}$	Activation variable for $(u,v) \in E_{res}$

3. SYSTEM MODEL AND PROBLEM STATEMENT

In this section we present the DSP application and resource model and define the operator replication and placement problem. For the sake of clarity, in Table 1 we summarize the notation used throughout the paper.

3.1 Abstract DSP Model

A DSP application can be represented at different levels of abstraction. The DSP *abstract model* defines the streams and their characteristics, along with the type, role and granularity of the stream processing elements. At this level, the DSP application can be regarded as a network of operators connected by streams. An operator is a self-contained processing element that carries out a specific operation (e.g., filtering, aggregation, merging) or something more complex

(e.g., POS-tagging), whereas a stream is an unbounded sequence of data (e.g., packet, tuple, file chunk).

A DSP abstract model can be represented as a labeled directed acyclic graph (DAG) $G_{dsp} = (V_{dsp}, E_{dsp})$, where the nodes in V_{dsp} represent the application operators as well as the data stream sources (i.e., nodes without incoming links) and sinks (i.e., nodes without outgoing links), and the links in E_{dsp} represent the streams, i.e., data flows, between nodes. Due to the difficulties of formalizing the non-functional attributes of an abstract operator, we characterize the operator with the non-functional attributes of a reference implementation on a reference architecture: Res_i, the amount of resources required for its execution; R_i, the operator latency (which accounts for the waiting time on the input queues as well as the execution time of a unit of data); C_i, the cost of deploying an instance of the operator. We characterize the stream exchanged from operator i to j, $(i,j) \in E_{dsp}$, with its average tuple rate $\lambda_{(i,j)}$ and average number of bytes per tuple $b_{(i,j)}$. To model load-dependent latency, we assume that the latency is function of λ_i, the operator input tuple rate, $R_i = R_i(\lambda_i)$, where $\lambda_i = \sum_{j \in V_{dsp}} \lambda_{(j,i)}$; without loss of generality, we also assume that R_i is an increasing function in λ_i. In this paper, we assume that Res_i is a scalar value, but our placement model can be easily extended to consider Res_i as a vector of required resources.

3.2 Resource Model

Computing and network resources can be represented as a labeled fully connected directed graph $G_{res} = (V_{res}, E_{res})$, where the set of nodes V_{res} represents the distributed computing resources, and the set of links E_{res} represents the *logical connectivity* between nodes. Observe that, at this level, links represent the logical links across the networks corresponding to the network paths between nodes (as determined by the network operator routing strategies). Each node $u \in V_{res}$ is characterized by Res_u, the amount of resources available at node u; S_u, the processing speed-up on a reference processor; and A_u, its availability, i.e., the probability that u is up and running. Each link $(u,v) \in E_{res}$, with $u, v \in V_{res}$ is characterized by: $d_{(u,v)}$, the network delay between node u and v; $A_{(u,v)}$, the link availability, i.e., the probability that the link between u and v is active; and, $C_{(u,v)}$, the cost per unit of data transmitted along the network path between u and v. This model considers also edges of the type (u,u); they capture network connectivity between operators placed in the same node u, and are considered as perfect links, i.e., always active with no network delay. We assume that the considered QoS attributes can be obtained by means of either active/passive measurements or with some network support (e.g., SDN).

3.3 Operator Replication and Placement

The DSP replication and placement problem consists in determining, for each operator $i \in V_{dsp}$, the number of replicas and where to deploy them on the computing nodes in V_{res}. Operators do not necessarily need to be mapped to a single node. In general, in order to improve performance, multiple instances of the same DSP operator can be instantiated over different computing nodes. The premise is that by partitioning the streams over multiple processing elements, we reduce the load of each processing element which in turn yields lower operator (and overall application) latency. Figure 1 represents a simple instance of the problem. Observe that a DSP operator cannot be usually placed on every node in V_{res}, because of physical (i.e., *pinned* operator) or other motivations (e.g., security, privacy). This observation allows us to consider for each operator $i \in V_{dsp}$ a subset of candidate resources $V^i_{res} \subseteq V_{res}$ where it can be deployed. For example, if sources and sinks ($I \subset V_{dsp}$) are external applications, their placement is fixed, that is $\forall i \in I, |V^i_{res}| = 1$.

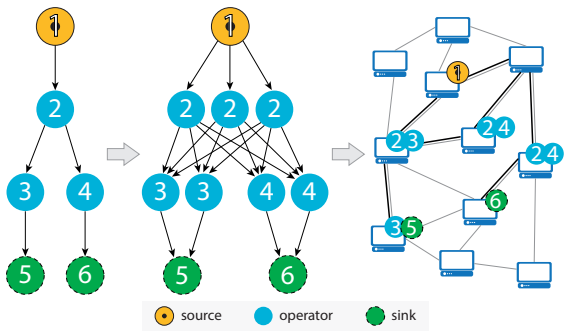


Figure 1: Replication of the application operators and their placement on the computing resources

The operator placement can be represented by a function *map* which maps an operator $i \in V_{dsp}$ to a multiset of computing nodes in V^i_{res}. We recur to multisets because a deployment can place multiple replicas of the same operator on the same computing node. For instance, $map(i) = \{u, u, v\}$, $i \in V_{dsp}$, $u, v \in V^i_{res}$, indicates that operator i deployment consists of 3 replicas, two of which on node u and one on node v. A multiset $\mathcal{X}$ over a set X, which we denote as $\mathcal{X} \sqsubset X$, is defined as a mapping $\mathcal{X} : X \to \mathbb{N}$ where for $x \in X$, $\mathcal{X}(x)$ denotes the multiplicity of x in $\mathcal{X}$. $x \in \mathcal{X}$ if and only if $\mathcal{X}(x) \geq 1$. The cardinality of a multiset $\mathcal{X}$, denoted $|\mathcal{X}|$, is defined by the number of elements in $\mathcal{X}$, that is $|\mathcal{X}| = \sum_{x \in \mathcal{X}} \mathcal{X}(x)$. Hereafter, without lack of generality, we will assume that in a deployment each operator $i \in V_{dsp}$ can be replicated at most k_i times.

We also find convenient to define the power multiset $\mathcal{P}(X)$ of a set X as the set of all multisets with elements taken from X and the subset $\mathcal{P}(X;k) \subset \mathcal{P}(X)$ of the multiset over X with cardinality no greater of k, that is $\mathcal{P}(X;k) = \{\mathcal{X} \in \mathcal{P}(X) | \sum_{x \in \mathcal{X}} \mathcal{X}(x) \leq k\}$.

4. OPTIMAL REPLICATION AND PLACEMENT MODEL

In this section we present our model for the ODRP problem. As the optimal solution depends on non-functional attributes, we first derive the expression for the different QoS metrics of interest and then present the ODRP formulation.

4.1 ODRP Variables

We model the ODRP problem with binary variables $x_{i,\mathcal{U}}$, $i \in V_{dsp}$ and $\mathcal{U} \sqsubset V^i_{res}$: $x_{i,\mathcal{U}} = 1$ if and only if the operator $map(i) = \mathcal{U}$, that is, i is replicated in $|\mathcal{U}|$ instances with exactly $\mathcal{U}(u)$ copies deployed in u, with $u \in \mathcal{U}$.

We also find convenient to consider binary variables associated to links, namely $y_{(i,j),(\mathcal{U},\mathcal{V})}$, $(i,j) \in E_{dsp}$, $\mathcal{U} \sqsubset V^i_{res}$, $\mathcal{V} \sqsubset V^j_{res}$, which denotes whether the data stream flowing from operator i to operator j traverses the network paths

from nodes in $\mathcal{U}$ to nodes in $\mathcal{V}$. By definition, we have $y_{(i,j),(\mathcal{U},\mathcal{V})} = x_{i,\mathcal{U}} \wedge x_{j,\mathcal{V}}$.

Finally, we also consider the variables z_u, $u \in V_{res}$ which denote whether at least an operator is deployed on node u and the variables $z_{(u,v)}$, $(u,v) \in E_{res}$ which denote whether a stream (or a portion of it) traverses the network path (u,v). By definition, we have $z_u = \vee_{i \in V_{dsp}, \mathcal{U} \in \mathcal{P}(V^i_{res};k_i)} x_{i,\mathcal{U}}$ and $z_{(u,v)} = \vee_{(i,j) \in E_{dsp}, \mathcal{U} \in \mathcal{P}(V^i_{res};k_i), \mathcal{V} \in \mathcal{P}(V^j_{res};k_j)} y_{(i,j),(\mathcal{U},\mathcal{V})}$.

For short, in the following we denote by $\boldsymbol{x}$ and $\boldsymbol{y}$ the placement vectors for nodes and edges, respectively, where $\boldsymbol{x} = \langle x_{i,\mathcal{U}} \rangle$, $\forall i \in V_{dsp}$, $\forall \mathcal{U} \in \mathcal{P}(V^i_{res};k_i)$, and $\boldsymbol{y} = \langle y_{(i,j),(\mathcal{U},\mathcal{V})} \rangle$, $\forall x_{i,\mathcal{U}}, x_{j,\mathcal{V}} \in \boldsymbol{x}$. Similarly, we denote by $\boldsymbol{z_V}$ and $\boldsymbol{z_E}$ the vectors $\boldsymbol{z_V} = \langle z_u \rangle$, $\forall u \in V_{res}$, and $\boldsymbol{z_E} = \langle z_{(u,v)} \rangle$, $\forall (u,v) \in E_{res}$.

4.2 Qos Metrics

4.2.1 Operator QoS Metrics

Let us first consider an operator in isolation. For each $i \in V_{dsp}$, the QoS of the operator deployment depends on the deployment $\mathcal{U}$. Let $R_{i,\mathcal{U}}$, $C_{i,\mathcal{U}}$, and $A_{i,\mathcal{U}}$ denote the maximum latency, the cost, and the availability of the deployment $\mathcal{U}$, respectively. We readily have:

$$R_{i,\mathcal{U}} = \max_{u \in \mathcal{U}} \frac{R_i(\frac{\lambda_i}{|\mathcal{U}|})}{S_u} \quad (1)$$

$$C_{i,\mathcal{U}} = \sum_{u \in \mathcal{U}} \mathcal{U}(u) C_i Res_i \quad (2)$$

$$A_{i,\mathcal{U}} = \prod_{u \in \mathcal{U}} A_u \quad (3)$$

under the assumption that the traffic is equally split among the different operator replicas.

4.2.2 Stream QoS Attributes

We now turn our attention to the QoS attributes related to a stream. For a stream (i,j), the QoS depends on the upstream and downstream operators' deployments $\mathcal{U}$ and $\mathcal{V}$. Let $d_{(i,j),(\mathcal{U},\mathcal{V})}$, $C_{(i,j),(\mathcal{U},\mathcal{V})}$, and $A_{(i,j),(\mathcal{U},\mathcal{V})}$ denote the maximum latency, the cost, and the availability of the deployments $\mathcal{U}$ and $\mathcal{V}$, respectively. We readily have:

$$d_{(i,j),(\mathcal{U},\mathcal{V})} = d_{(\mathcal{U},\mathcal{V})} = \max_{u \in \mathcal{U}, v \in \mathcal{V}} d_{(u,v)} \quad (4)$$

$$C_{(i,j),(\mathcal{U},\mathcal{V})} = \sum_{u \in \mathcal{U}, v \in \mathcal{V}} \lambda_{(i,j),(\mathcal{U},\mathcal{V})} C_{u,v} \quad (5)$$

$$A_{i,(\mathcal{U},\mathcal{V})} = \prod_{u \in \mathcal{U}, v \in \mathcal{V}} A_{(u,v)} \quad (6)$$

where

$$\lambda_{(i,j),(\mathcal{U},\mathcal{V})} = \frac{\lambda_{(i,j)}}{|\mathcal{U}||\mathcal{V}|} \quad u \in \mathcal{U}, v \in \mathcal{V} \quad (7)$$

is the amount of stream (i,j) traffic exchanged between two operator replicas under the deployments $\mathcal{U}$ and $\mathcal{V}$.

4.2.3 DSP Application QoS Metrics

We consider both user-oriented and system-oriented QoS metrics, such as application response time and availability for the former, and network related metrics for the latter.

Response Time: For a DSP application, with data flowing from several sources to several destinations, there is no unique definition of response time. In the following, we consider as response time R the worst end-to-end delay from a source to a sink. Given this definition, we have that:

$$R = \max_{p \in \pi_{G_{dsp}}} R_p \quad (8)$$

being R_p the delay along path p and $\pi_{G_{dsp}}$ the set of all source-sink paths in G_{dsp}. Given a placement vector $\boldsymbol{x}$ (and resulting $\boldsymbol{y}$) and a path $p = (i_1, i_2, \ldots, i_{n_p})$, we have $R = R(\boldsymbol{x},\boldsymbol{y}) = \max_{p \in \pi_{G_{dsp}}} R_p(\boldsymbol{x},\boldsymbol{y})$ with $R_p(\boldsymbol{x},\boldsymbol{y})$ defined as:

$$R_p(\boldsymbol{x},\boldsymbol{y}) = \sum_{k=1}^{n_p} R_{i_k}(\boldsymbol{x}) + \sum_{k=1}^{n_p-1} D_{(i_k,i_{k+1})}(\boldsymbol{y}) \quad (9)$$

where

$$R_i(\boldsymbol{x}) = \sum_{\mathcal{U} \in \mathcal{P}(V^i_{res};k_i)} R_{i,\mathcal{U}} x_{i,\mathcal{U}} \quad (10)$$

$$D_{(i,j)}(\boldsymbol{y}) = \sum_{\substack{\mathcal{U} \in \mathcal{P}(V^i_{res};k_i) \\ \mathcal{V} \in \mathcal{P}(V^j_{res};k_j)}} d_{(\mathcal{U},\mathcal{V})} y_{(i,j),(\mathcal{U},\mathcal{V})} \quad (11)$$

denote respectively the execution time of operator i when deployed over the multiset $\mathcal{U}$ and the worst case network delay for transferring data from i to j when the two operators are mapped over $\mathcal{U}$ and $\mathcal{V}$, respectively.

Cost: We define the cost C of the DSP application as the monetary cost of all the computing resources and paths involved in the processing and transmission of the application data streams. We have:

$$C(\boldsymbol{x},\boldsymbol{y}) = \sum_{i \in V_{dsp}} C_i(\boldsymbol{x}) + \sum_{(i,j) \in E_{dsp}} C_{(i,j)}(\boldsymbol{y}) \quad (12)$$

where

$$C_i(\boldsymbol{x}) = \sum_{\mathcal{U} \in \mathcal{P}(V^i_{res};k_i)} C_{i,\mathcal{U}} x_{i,\mathcal{U}} \quad (13)$$

$$C_{(i,j)}(\boldsymbol{y}) = \sum_{\substack{\mathcal{U} \in \mathcal{P}(V^i_{res};k_i) \\ \mathcal{V} \in \mathcal{P}(V^j_{res};k_j)}} C_{(i,j),(\mathcal{U},\mathcal{V})} y_{(i,j),(\mathcal{U},\mathcal{V})} \quad (14)$$

Availability: We define the application availability A as the availability of all the nodes and paths involved in the processing and transmission of the application data streams. For the sake of simplicity, we assume the availability of the different components to be independent. We acknowledge that independence does not hold true in general and that a more detailed model is needed to capture the dependency relationship among logical components sharing physical nodes and networks links; we postpone it to future work. With the independence assumption, we readily have:

$$A(\boldsymbol{z_V},\boldsymbol{z_E}) = \prod_{u \in V_{res}: z_u = 1} A_u z_u \cdot \prod_{(u,v) \in E_{res}: z_{(u,v)} = 1} A_{(u,v)} z_{(u,v)} \quad (15)$$

To obtain a linear expression, we consider the logarithm of the availability, obtaining:

$$\log A(\boldsymbol{z_V},\boldsymbol{z_E}) = \sum_{u \in V_{res}} a_u z_u + \sum_{(u,v) \in E_{res}} a_{(u,v)} z_{(u,v)} \quad (16)$$

where $a_u = \log A_u$ and $a_{(u,v)} = \log A_{(u,v)}$. It is worth observing that in (16) we can take the summation over all $u \in V_{res}$ and $(u,v) \in E_{res}$ since the terms not appearing in (15) are those corresponding to $z_u = 0$ or $z_{(u,v)} = 0$, which do not affect the summation in (16).

Network Related QoS Metrics: In the DSP literature, several alternative network-aware metrics have been defined, including the inter-node traffic T [1], the network usage N [16], and an approximation of the elastic energy EE [13]. Let $Z(\boldsymbol{y})$, $Z = T|N|EE$, denote the QoS attribute of the DSP application under the placement policy $\boldsymbol{y}$, we have:

$$Z(\boldsymbol{y}) = \sum_{(i,j)\in E_{dsp}} Z_{(i,j)}(\boldsymbol{y}) \tag{17}$$

where $Z_{(i,j)}(\boldsymbol{y})$ is defined as follows.

The *inter-node traffic* T is the overall amount of data exchanged per time unit between operators placed on different nodes. Therefore, using the placement policy $\boldsymbol{y}$, the stream $(i,j) \in E_{dsp}$ generates an inter-node traffic equals to:

$$T_{(i,j)}(\boldsymbol{y}) = \sum_{\substack{u\in\mathcal{U},v\in\mathcal{V},u\neq v\\ \mathcal{U}\in\mathcal{P}(V^i_{res};k_i)\\ \mathcal{V}\in\mathcal{P}(V^j_{res};k_j)}} b_{(i,j)}\lambda_{(i,j),(\mathcal{U},\mathcal{V})}y_{(i,j),(\mathcal{U},\mathcal{V})} \tag{18}$$

The *network usage* N is the amount of data that traverses the network at a given time; therefore, the stream $(i,j) \in E_{dsp}$ imposes a load expressed by:

$$N_{(i,j)}(\boldsymbol{y}) = \sum_{\substack{u\in\mathcal{U},v\in\mathcal{V},u\neq v\\ \mathcal{U}\in\mathcal{P}(V^i_{res};k_i)\\ \mathcal{V}\in\mathcal{P}(V^j_{res};k_j)}} b_{(i,j)}\lambda_{(i,j),(\mathcal{U},\mathcal{V})}d_{(u,v)}y_{(i,j),(\mathcal{U},\mathcal{V})} \tag{19}$$

where $d_{(u,v)}$ is the network delay among nodes $u, v \in V_{res}$, with $u \neq v$.

In their paper [13], Pietzuch et al. indirectly minimize the network usage through the minimization of the elastic energy, which results from the equivalent system of springs that represents the application. Basically, their solution minimizes the amount of data that traverses each link weighted by the latency of the link itself. Hence, the stream $(i,j) \in E_{dsp}$ contributes to the elastic energy of the system with:

$$EE_{(i,j)}(\boldsymbol{y}) = \sum_{\substack{u\in\mathcal{U},v\in\mathcal{V},u\neq v\\ \mathcal{U}\in\mathcal{P}(V^i_{res};k_i)\\ \mathcal{V}\in\mathcal{P}(V^j_{res};k_j)}} b_{(i,j)}\lambda_{(i,j),(\mathcal{U},\mathcal{V})}d^2_{(u,v)}y_{(i,j),(\mathcal{U},\mathcal{V})} \tag{20}$$

Observe that, in all cases, $Z(\boldsymbol{y})$ is a linear function of $\boldsymbol{y}$.

4.3 ODRP Formulation

Depending on the usage scenario, a DSP replication and placement strategy could be aimed at optimizing different, possibly conflicting, QoS attributes. To this end, we use the Simple Additive Weighting (SAW) technique [17] to define the utility function $F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z_V}, \boldsymbol{z_E})$ as a weighted sum of the normalized QoS attributes of the application, as follows:

$$F(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z_V},\boldsymbol{z_E}) = w_r\frac{R_{\max}-R(\boldsymbol{x},\boldsymbol{y})}{R_{\max}-R_{\min}} + w_a\frac{\log A(\boldsymbol{z_V},\boldsymbol{z_E})-\log A_{\min}}{\log A_{\max}-\log A_{\min}} + w_c\frac{C_{\max}-C(\boldsymbol{x},\boldsymbol{y})}{C_{\max}-C_{\min}} + w_z\frac{Z_{\max}-Z(\boldsymbol{x},\boldsymbol{y})}{Z_{\max}-Z_{\min}} \tag{21}$$

where $w_r,\ w_a,\ w_c,\ w_z \geq 0$, $w_r + w_a + w_c + w_z = 1$, are weights associated to the different QoS attributes. $R_{\max}$ ($R_{\min}$), $A_{\max}$ ($A_{\min}$), $C_{\max}$ ($C_{\min}$), and $Z_{\max}$ ($Z_{\min}$) denote, respectively, the maximum (minimum) value for the overall expected response time, availability, cost and network related metric. Observe that after normalization, each metric ranges in the interval $[0,1]$.

We formulate the ODRP problem as an Integer Linear Programming (ILP) model as follows:

$$\max_{\boldsymbol{x},\boldsymbol{y},r} F'(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z_V},\boldsymbol{z_E},r)$$

subject to:

$$r \geq \sum_{\substack{k=1,\dots,n_p\\ \mathcal{U}\in\mathcal{P}(V^i_{res};k_i)}} R_{i,\mathcal{U}}x_{i,\mathcal{U}} + \sum_{\substack{k=1,\dots,n_p=1\\ \mathcal{U}\in\mathcal{P}(V^i_{res};k_i)\\ \mathcal{V}\in\mathcal{P}(V^j_{res};k_j)}} d_{(\mathcal{U},\mathcal{V})}y_{(i,j),(\mathcal{U},\mathcal{V})} \qquad \forall p\in\pi_G \tag{22}$$

$$Res_u \geq \sum_{\substack{i\in V_{dsp}\\ \mathcal{U}\in\mathcal{P}(V^i_{res})}} \mathcal{U}(u)Res_i x_{i,\mathcal{U}} \qquad \forall u\in V_{res} \tag{23}$$

$$z_u \geq \frac{\sum_{i\in V_{dsp}} \sum_{\mathcal{U}\in\mathcal{P}(V^i_{res};k_i)} x_{i,\mathcal{U}}}{M} \qquad u\in V_{res} \tag{24}$$

$$z_{(u,v)} \geq \frac{\sum_{(i,j)\in E_{dsp}} \sum_{\substack{\mathcal{U}\in\mathcal{P}(V^i_{res};k_i)\\ \mathcal{V}\in\mathcal{P}(V^j_{res};k_j)}} y_{(i,j),(\mathcal{U},\mathcal{V})}}{N} \qquad (u,v)\in E_{res} \tag{25}$$

$$\sum_{\mathcal{U}\in\mathcal{P}(V^i_{res};k_i)} x_{i,\mathcal{U}} = 1 \qquad \forall i\in V_{dsp} \tag{26}$$

$$x_{i,\mathcal{U}} = \sum_{\mathcal{V}\in\mathcal{P}(V^j_{res};k_j)} y_{(i,j),(\mathcal{U},\mathcal{V})} \qquad \begin{matrix}\forall(i,j)\in E_{dsp},\\ \mathcal{U}\in\mathcal{P}(V^i_{res};k_i)\end{matrix} \tag{27}$$

$$x_{j,\mathcal{V}} = \sum_{\mathcal{U}\in\mathcal{P}(V^i_{res};k_u)} y_{(i,j),(\mathcal{U},\mathcal{V})} \qquad \begin{matrix}\forall(i,j)\in E_{dsp},\\ \mathcal{V}\in\mathcal{P}(V^j_{res};k_j)\end{matrix} \tag{28}$$

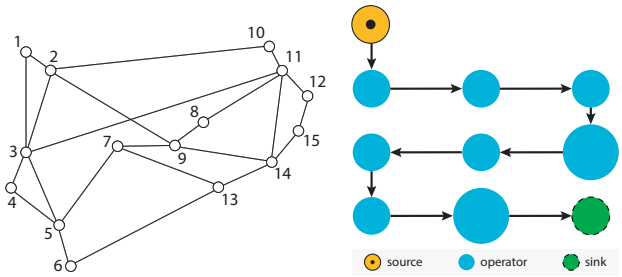
$$x_{i,\mathcal{U}} \in \{0,1\} \qquad \begin{matrix}\forall i\in V_{dsp},\\ \mathcal{U}\in\mathcal{P}(V^i_{res};k_i)\end{matrix} \tag{29}$$

$$y_{(i,j),(\mathcal{U},\mathcal{V})} \in \{0,1\} \qquad \begin{matrix}\forall(i,j)\in E_{dsp},\\ \mathcal{U}\in\mathcal{P}(V^i_{res};k_i)\\ \mathcal{V}\in\mathcal{P}(V^j_{res};k_j)\end{matrix} \tag{30}$$

$$z_u \in \{0,1\} \qquad \forall u\in V_{res} \tag{31}$$

$$z_{(u,v)} \in \{0,1\} \qquad \forall (u,v)\in E_{res} \tag{32}$$

In the problem formulation we use the objective function $F'(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z_V},\boldsymbol{z_E},r)$ that is obtained from $F(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z_V},\boldsymbol{z_E})$ by replacing $R(\boldsymbol{x},\boldsymbol{y})$ with the auxiliary variable r, which represents the application response time in the optimization problem, in order to obtain a linear objective function. Observe that, indeed, while F is nonlinear in $\boldsymbol{x}$, $\boldsymbol{y}$ since $R(\boldsymbol{x},\boldsymbol{y}) = \max_{p\in\pi_{G_{dsp}}} R_p(\boldsymbol{x},\boldsymbol{y})$ is a nonlinear term, F' is linear in r as well as in $\boldsymbol{x}$ and $\boldsymbol{y}$. Equation (22) follows from (8)–(11). Since r must be larger or equal than the response time of any path and, at the optimum, r is minimized, $r = \max_{p\in\pi_{G_{dsp}}} R_p(\boldsymbol{x},\boldsymbol{y}) = R(\boldsymbol{x},\boldsymbol{y})$. The constraint (23) limits the placement of operators on a node $u \in V_{res}$ ac-



(a) Subset of ANSNET nodes (b) Reference application

Figure 2: Reference infrastructure and DSP application

cording to its available resources. Constraints (24) and (25) are the activation constraints for the variable z_u and $z_{(u,v)}$, respectively, with M and N large constants. Equation (26) guarantees that each operator $i \in V_{dsp}$ is placed on one and only one node $u \in V^i_{res}$. Finally, constraints (27)–(28) model the logical AND between the placement variables, that is, $y_{(i,j),(u,v)} = x_{i,u} \wedge x_{j,v}$.

THEOREM 1. *The ODRP problem is an NP-hard problem.*

PROOF. It is sufficient to observe that the Optimal DSP Replication and Placement problem is a generalization of the Optimal DSP Placement problem we presented in [2], which has been shown to be NP-hard. □

5. EXPERIMENTAL RESULTS

We evaluate the ODRP model through a set of numerical experiments that aim at demonstrating the flexibility of the formulation and the benefits that derive from the joint optimization of operators placement and replication. We analyze in Section 5.2 how the ODRP formulation allows us to consider the optimization of several QoS metrics, such as response time, cost, availability, and inter-node traffic. Then, in Section 5.3 we show the impact of replication when the DSP application is subject to an increasing load.

5.1 Experimental Setup

The ODRP model allows us to define the operators placement and replication by optimizing different QoS attributes, whose importance depends on the utilization scenario.

We solve the ILP problem using CPLEX© (version 12.6.2) on an Amazon EC2 virtual machine (c4.xlarge with 4 vCPU and 7.5 GB RAM). In the experiments, G_{res} models a portion of ANSNET, a geographically distributed network where 15 computing nodes are interconnected with non-negligible network delays. Within this network, we assume that a logical link $(u, v) \in E_{res}$ between any two computing resources $u, v \in V_{res}$ always exists; each logic link results by the underlying physical network paths, which are represented in Figure 2a, and a shortest-path routing strategy.

As reference application, we consider the topology represented in Figure 2b, which is made of a sequence of 10 operators, where the first and last one are respectively the source and sink. If not otherwise specified, the application includes two CPU-intensive operators, represented with a bigger shape, which require twice of the computing resources Res_i and have a service time five times longer than the other operators. We summarize the application and infrastructure configuration parameters in Table 2.

Table 2: Parameters of the experimental setup

Processing and network resources

Parameter	Value	Parameter	Value
$\|V_{res}\|$	15	$A_{(u,v)}$	100 %
A_u	Uniform in [97, 99.99999] %	$C_{(u,v)}$	0.02
Res_u	5	avg $d_{(u,v)}$	32 ms
S_u	1.0		

Application

Parameter	Value	Parameter	Value
$\|V_{dsp}\|$	10	Res_i	1
C_i	1	μ_i	0.02 ms^{-1}
$R_i(\lambda_i/\|\mathcal{U}\|)$	$\frac{1}{\mu_i - \lambda_i/\|\mathcal{U}\|}$ ms	$b_{(i,j)}$	1500 B/tuple
λ_i	14 tuples/s		

Normalization factors for the ODRP utility function

Parameter	Value	Parameter	Value
$R_{\min}$	1230 ms	$R_{\max}$	2810 ms
$A_{\min}$	85 %	$A_{\max}$	98.8 %
$C_{\min}$	11.0	$C_{\max}$	25.0
$Z_{\min}$	8.0 KB/s	$Z_{\max}$	40.0 KB/s

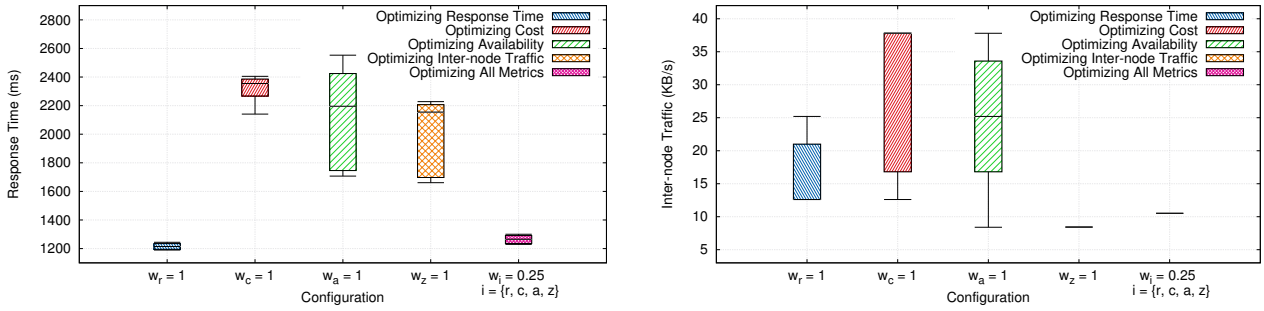
We rely on ODRP to define the optimal placement and replication of each operator but the source and sink, which are placed a-priori. In the experiments, we assume that each operator can be replicated at most twice (i.e., $k_i = 2, \forall i$), while the source and sink are not replicated. In the ODRP model, we obtain the response time R_i of the operator i subject to the incoming load $\lambda_i/|\mathcal{U}|$ by modeling the underlying computing node as an M/M/1 queue.

5.2 Joint Optimization of Operator Placement and Replication

This first experiment evaluates the effect on QoS metrics of different optimization objectives. We first compute the replication and placement solution by optimizing a single QoS metric. For example, to optimize the response time we set the weights as $w_r = 1$, $w_a = w_c = w_z = 0$. Then, we optimize the multi-objective function by uniformly weighting each metrics contribution (i.e., $w_r = w_a = w_c = w_z = 0.25$); we report in Table 2 the normalization factors used in Equation (21). Since the position of data source and sink affects the optimal solution, we execute a run for each configuration that results by pinning these components on every pair of distinct nodes. Figure 3 presents the results in term of the different QoS metrics. For each optimization objective, a boxplot represents the performance distribution that results from the different locations of data source and sink. Each boxplot reports the minimum value, the 5th percentile, the median, the 95th percentile, and the maximum value.

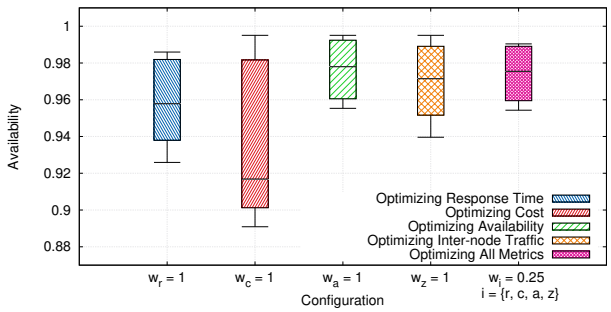
When ODRP optimizes the response time R, the solution achieves the minimum achievable response time, as shown in Figure 3a, but also requires that all the operators are replicated, as shown in Figure 3d. Since the cost of running a configuration is directly proportional to the total number of operator instances, this is also the most expensive solution. We can also observe from Figure 3b that the inter-node traffic is indirectly minimized, because transmitting data over the network rather than locally introduces network delays that penalize the response time.

When ODRP optimizes the cost C, the placement solution tries to use less computing resources as possible, therefore no operator is replicated as shown in Figure 3d. This strategy penalizes the response time which almost doubles

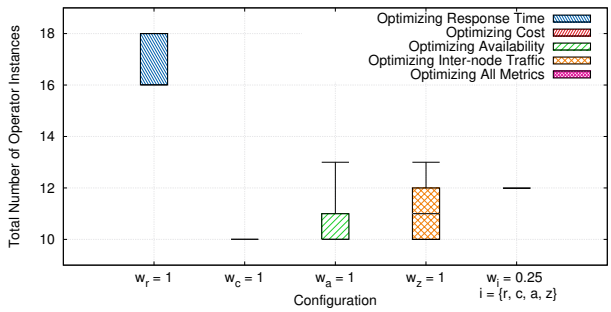


(a) Application response time

(b) Inter-node traffic

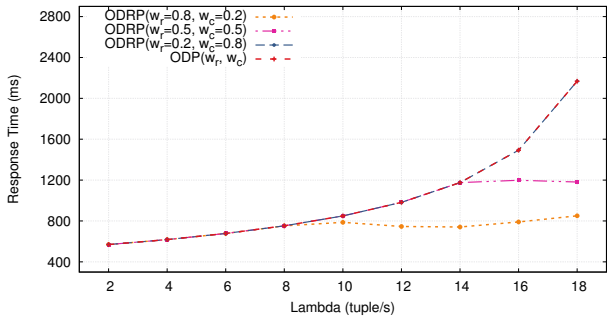


(c) Application availability

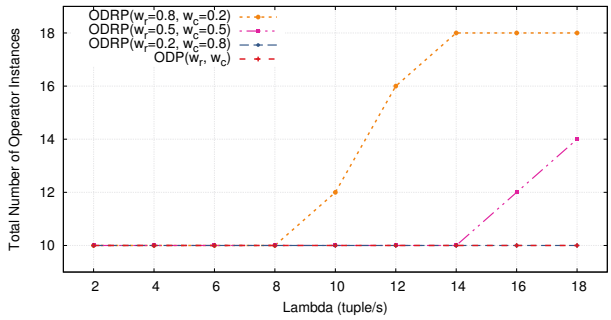


(d) Total number of operator instances

Figure 3: Impact of different optimization objectives on the QoS metrics



(a) Response time



(b) Total number of operator instances

Figure 4: Impact of replication on the application performance

with respect to the minimum value, as shown in Figure 3a. Nothing can be concluded about the application availability (Figure 3c) and inter-node traffic (Figure 3b): since these metrics are not optimized, there is a set of equally optimal solutions that differ each other only in the operator placement on the same set of computing resources.

At a first sight, the results obtained when ODRP optimizes the application availability seem to be unexpected, because, as shown in Figure 3c, the application availability is not maximized. However, this behavior can be easily explained recalling that the data source and sink are placed a-priori on each pair of computing nodes, so their availability drives the overall application availability. In some cases ODRP increases the number of operator instances to improve the application availability: from Figure 3d, we can see that the maximum number of instances in this configuration is 13, i.e., 3 operators are replicated with degree 2.

When ODRP optimizes the inter-node traffic, the solution tries to co-locate the operator instances on the least number of nodes, so to reduce the amount of data transmitted over the network (see Figure 3b). An interesting behavior is highlighted by Figure 3d: sometimes ODRP takes advantage of replication in order to reduce the amount of data transmitted on the network; this behavior is analytically explained by Equation (7).

When ODRP optimizes all the considered QoS metrics, the resulting placement and replication solution has a response time (Figure 3a) and an inter-node traffic (Figure 3b) which are very close to the values achievable when optimizing a single-objective function. The total number of operator instances is always equal to 12, as shown in Figure 3d, therefore only two operators are replicated. Recalling the topology of our application, we can readily identify that these two replicated operators are the application bottlenecks.

5.3 Impact of Replication

In the last experiment we want to investigate the replication benefits when the application is subject to an increasing traffic load. Differently from the previous experiment, now each operator emits only 95 % of the incoming data; in this way, every operator imposes a different load which decreases from the source to the sink. We compare the performance achieved by ODRP with ODP, which optimizes only the operator placement [2]. Note that ODP is a special case of ODRP where $k_i = 1, \forall i \in V_{dsp}$. Both the models are solved with the following configurations of weights: $(w_r = 0.8, w_c = 0.2)$, $(w_r = 0.5, w_c = 0.5)$, and $(w_r = 0.2, w_c = 0.8)$. The other weights are set to 0. In Figure 4 we indicate with "ODP(w_r, w_c)" the curves obtained by ODP under the different weight configurations, which overlap one another. Note also that "ODP(w_r, w_c)" overlaps with "ODRP$(w_r = 0.2, w_c = 0.8)$".

During the experiment, the incoming load increases from 2 to 18 tuples/s, therefore the bottleneck operator imposes a load on the underlying computing node that ranges from 10% to 90% of its capacity. Although ODP cannot replicate the operators, it can find the optimal placement that minimizes the response time. As shown in Figure 4a, when the system becomes overloaded, i.e., after 14 tuples/s, the response time grows quickly, up to become 4 times higher than the case of medium load. A similar result is obtained when ODRP is configured with $(w_r = 0.2, w_c = 0.8)$: being the application cost much more important than the response time, ODRP tries not to replicate the operators. The opposite result is achieved when ODRP is configured with $(w_r = 0.8, w_c = 0.2)$, because replication is exploited to reduce the application response time. Figure 4b shows the number of operator instances. We can observe that, when the incoming load exceeds 8 tuples/s, ODRP with $(w_r = 0.8, w_c = 0.2)$ starts replicating the bottleneck operators one by one up to 18 instances, i.e., when every operator is replicated. As shown in Figure 4a, this strategy is beneficial for the response time, which does not increase more than 1.5 times with respect to the case of lightly loaded system.

6. CONCLUSIONS

In this paper, we presented ODRP, an ILP formulation that jointly optimizes the replication and placement of DSP applications. ODRP is a general and flexible model that can take into account the heterogeneity of computing and networking resources and can be conveniently configured to optimize different QoS metrics, whose importance depends on the application scenario. The experimental evaluation has validated the proposed model and has shown the benefits of a joint optimization of the operators placement and their replication on the application performance.

As future work, we plan to develop efficient heuristics to deal with large problem instances for the initial application replication and placement. Moreover, we plan to extend our model in order to support on-line reconfigurations, where the operator deployment needs to be adapted at run-time to properly handle input streams or execution environments with continuously changing properties.

7. ACKNOWLEDGMENTS

This publication is supported by COST Action ACROSS (IC1304).

8. REFERENCES

[1] L. Aniello, R. Baldoni, and L. Querzoni. Adaptive online scheduling in Storm. In *Proc. of ACM DEBS '13*, pages 207–218, 2013.

[2] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli. Optimal operator placement for distributed stream processing applications. In *Proc. of ACM DEBS '16*, pages 69–80, 2016.

[3] V. Cardellini, M. Nardelli, and D. Luzi. Elastic stateful stream processing in Storm. In *Proc. of HPCS '16*, pages 583–590. IEEE, 2016.

[4] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. of OSDI '04*. USENIX Association, 2004.

[5] R. Eidenbenz and T. Locher. Task allocation for distributed stream processing. In *Proc. of IEEE INFOCOM '16*, 2016.

[6] T. Z. J. Fu, J. Ding, R. T. B. Ma, M. Winslett, et al. DRS: Dynamic resource scheduling for real-time analytics over fast streams. In *Proc. of IEEE ICDCS 2015*, pages 411–420, 2015.

[7] B. Gedik, S. Schneider, M. Hirzel, and K.-L. Wu. Elastic scaling for data stream processing. *IEEE Trans. Parallel Distrib. Syst.*, 25(6):1447–1463, 2014.

[8] T. Heinze, Z. Jerzak, G. Hackenbroich, and C. Fetzer. Latency-aware elastic scaling for distributed data stream processing systems. In *Proc. of ACM DEBS '14*, pages 13–22, 2014.

[9] T. Heinze, V. Pappalardo, Z. Jerzak, and C. Fetzer. Auto-scaling techniques for elastic data stream processing. In *Proc. of IEEE ICDEW '14*, pages 296–302, 2014.

[10] T. Heinze, L. Roediger, A. Meister, Y. Ji, et al. Online parameter optimization for elastic data stream processing. In *Proc. of ACM SoCC '15*, pages 276–287, 2015.

[11] B. Lohrmann, P. Janacik, and O. Kao. Elastic stream processing with latency guarantees. In *Proc. of IEEE ICDCS '15*, pages 399–410, 2015.

[12] G. Mencagli. A game-theoretic approach for elastic distributed data stream processing. *ACM Trans. on Autonomous and Adaptive Systems*, 11(2), 2016.

[13] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, et al. Network-aware operator placement for stream-processing systems. In *Proc. of IEEE ICDE '06*, 2006.

[14] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things J.*, 3(5):637–646, Oct. 2016.

[15] C. Thoma, A. Labrinidis, and A. Lee. Automated operator placement in distributed data stream management systems subject to user constraints. In *Proc. of IEEE ICDEW '14*, pages 310–316, 2014.

[16] J. Xu, Z. Chen, J. Tang, and S. Su. T-Storm: traffic-aware online scheduling in Storm. In *Proc. of IEEE ICDCS '14*, pages 535–544, 2014.

[17] K. P. Yoon and C.-L. Hwang. *Multiple Attribute Decision Making: an Introduction*. Sage Pubns, 1995.