Optimal Operator Placement for Distributed Stream Processing 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

Data Stream Processing (DSP) applications are widely used to timely extract information from distributed data sources, such as sensing devices, monitoring stations, and social networks. To successfully handle this ever increasing amount of data, recent trends investigate the possibility of exploiting decentralized computational resources (e.g., Fog computing) to define the applications placement. Several placement policies have been proposed in the literature, but they are based on different assumptions and optimization goals and, as such, they are not completely comparable to each other.

In this paper we study the placement problem for distributed DSP applications. Our contributions are twofold. We provide a general formulation of the optimal DSP placement (for short, ODP) as an Integer Linear Programming problem which takes explicitly into account the heterogeneity of computing and networking resources and which encompasses - as special cases - the different solutions proposed in the literature. We present an ODP-based scheduler for the Apache Storm DSP framework. This allows us to compare some well-known centralized and decentralized placement solutions. We also extensively analyze the ODP scalability with respect to various parameter settings.

CCS Concepts

•**Software and its engineering** → **Distributed systems organizing principles;**

Keywords

Distributed data stream processing, Optimal placement, QoS

1. INTRODUCTION

The advent of the Big Data era and the diffusion of the Cloud computing paradigm have renewed the interest in Data Stream Processing (DSP) applications [14], which can continuously collect and process data generated by an increasing number of sensing devices, to timely extract valuable information about many fundamental aspects of the environment we live in.

The emerging scenario pushes DSP systems to a whole new performance level. Strict quality requirements, great volumes of data, and high production rate exacerbate the need of an efficient usage of the underlying infrastructure.

To date, DSP applications are typically deployed on large-scale and centralized (Cloud) data centers that are often distant from data sources. However, as data increase in size, pushing them towards the Internet core could cause excessive stress for the network infrastructure and also introduce excessive delays. A solution to improve scalability and reduce network latency lies in taking advantage of the ever increasing presence of near-edge/Fog Cloud computing resources [4] and decentralizing the DSP application, by moving the computation to the edges of the network close to data sources. Nevertheless, the use of a diffused infrastructure poses new challenges that include network and system heterogeneity, geographic distribution as well as non-negligible network latencies among distinct nodes processing different parts of a DSP application. This latter aspect in particular could have a strong impact on DSP applications for latency-sensitive domains (e.g., [5]).

For this scenario, a relevant problem thus consists in determining, within a set of available distributed computing nodes, the nodes that should host and execute each operator of a DSP application, with the goal of optimizing the Quality of Service (QoS) attributes of the application. This problem is known as the *operator placement problem* (or scheduling problem). Several placement policies have been already proposed in literature (e.g., [1, 19, 21, 25, 28]), which are characterized by different assumptions and optimization goals, as pointed out in [17]. However, there is no general formulation of the placement problem, which makes difficult to analyse and compare the different solutions.

In this paper, we propose Optimal DSP Placement (**ODP**), a unified general formulation of the operator placement problem for distributed and networked DSP applications, which takes into account the heterogeneity of application requirements and infrastructural resources. Differently from prior approaches (e.g., [10, 16, 23, 29]), ODP provides a general modeling framework that can be easily extended to include new constraints and QoS attributes, and that at the same

DEBS '16, June 20-24, 2016, Irvine, CA, USA

DOI: http://dx.doi.org/10.1145/2933267.2933312

time provides a benchmark against which other centralized and decentralized placement algorithms can be compared.

The main contributions of our paper are as follows.

- We model the ODP problem as an Integer Linear Programming (ILP) problem (Sections 3 and 4), which can be used to optimize different QoS metrics. We first propose a basic formulation that considers only user-oriented metrics, i.e., the application end-to-end latency and availability. Then, to show how easily ODP can be extended, we include network-related QoS metrics, considering the exchanged data rate between the application operators as additional metric (Section 5). Similarly, other metrics (e.g., monetary cost, memory) can be considered.
- We present a prototype scheduler for Apache Storm, a well-known open source DSP framework, where the DSP application operators are placed according to the ODP solution (Section 6).
- Using Storm, we show how ODP can optimize different user-oriented QoS metrics (Section 7.1) and compare the performance achieved by some centralized and decentralized placement policies (i.e., [19, 21, 25]) to that obtained by ODP (Section 7.2). To this end, we have also implemented into Storm the centralized placement policy in [25] and the decentralized one in [21].
- We extensively evaluate the computational cost of solving ODP, under different configurations of application requirements and resource capabilities, and examine two simple strategies to reduce the resolution time (Section 7.3).

2. RELATED WORK

The DSP placement problem has been widely investigated in the literature under different modeling assumptions and optimization goals. Due to the NP-hardness of the placement problem, several heuristics have been proposed.

Placement Problem Formulation.

Among the first proposals, Zhu et al. [29] studied a placement problem which accounts for computational and communicational delays. However, they assume that a resource node can host at most a single operator; we consider this hypothesis not realistic in today's DSP systems, therefore our formulation enables the co-location of operators on a resource node, according to the node computational capacity.

In [10], Eidenbenz et al. analyze the placement problem for a subset of DSP application topologies, i.e., serial-parallel decomposable graphs. This allows them to exploit strong theoretical foundations and propose an approximation algorithm, which, however, can allocate operators only on resources with uniform capacity.

Thoma et al. [23] model the relationship between operators and resources, improving the expressiveness of the user constraints to include co-location, upstream/downstream, and attribute or tag-based constraints. We focus only on global QoS metrics and do not investigate this issue, however the related user constraints can be easily integrated within our placement model.

It is worth mentioning the work by Huang et al. [15], where the authors elegantly rearrange the Multi-Constrained Optimal Path problem to model and solve the service composition problem. Nevertheless, their model allows only to express constraints on communication links but not on computing resources.

Stanoi et al. [22] focus on maximizing the rate of the input streams that the DSP system can support, acting on both the order of operators and the placement on the resource nodes. We do not consider operator re-ordering as possible.

Unlike all the above approaches, we model the placement problem for composite and networked applications that, as DSP applications, can be represented by a directed acyclic graph, with a holistic vision of both computing and networking resources. The proposed formulation considers QoS attributes of both the applications and resources, and is flexible enough to accommodate new QoS metrics. Thanks to the adjustment of suitable knobs, the meaning of "optimal placement" can change according to the application context. As a consequence, ODP provides a general framework for DSP QoS optimization and comparison of different approaches.

Similarly to our approach, Benoit et al. [3] use an ILP problem to represent the placement problem for in-network stream-processing applications, which are a slightly different kind of applications, with the topology restricted to a binary tree of operators.

Heuristic Approaches.

Different heuristic approaches aim at optimizing a diversity of utility functions, such as to minimize the application end-to-end latency (e.g., [2, 7]), the inter-node traffic (e.g., [1, 25, 28]), and the network usage (e.g., [19, 21]). Backman et al. [2] and Chatzistergiou et al. [7] propose two different heuristic approaches to partition and assign group of operators while minimizing the application latency. Fischer et al. [12] use a graph partitioning technique to optimize the amount of data sent between nodes.

Huang et al. [16] model the relationship between the operator execution time and the amount of residual computing capacity on a resource node and propose a best-fit heuristic that aims at minimizing the network usage.

Relying on the best-fit meta-heuristic, Aniello et al. [1] and Xu et al. [25] propose centralized algorithms that assign operators with the goal of reducing the inter-node traffic exchanged on the network during the application execution. Specifically, the solution in [1] co-locates on the same node the pairs of operators that communicate with high data rate, whereas the proposal in [25] assigns each operator in descending order of incoming and outgoing traffic, minimizing the inter-node traffic. The same metric is considered by the decentralized solution presented by Zhou et al. [28], which finds the placement while balancing the load among computing nodes.

Pietzuch et al. [19] and Rizou et al. [21] minimize the network usage, that is the amount of data that traverses the network at a given instant. Both the solutions propose a decentralized placement algorithm that exploits a latency space as a search space to find the optimal placement solution in a completely distributed way. In [19] the application is modeled as an equivalent system of springs: operators are massless bodies tied together by springs that represent the exchanged streams. The stream data-rate and network latency determine the stretching of the springs. The network usage is indirectly minimized by finding the assignment that minimizes the overall elastic energy of this equivalent system. Differently, Rizou et al. [21] exploit the mathemati-

cal properties of the network usage function: if each operator iteratively finds its local optimal placement, the optimal placement for the application is achieved.

Although all these solutions are intended for solving the same problem, they differ from each other on their own assumptions and optimization goals [17], leading to strategies not fully comparable one to each other.

DSP Frameworks.

Apache Storm [24] is a DSP framework that provides an abstraction layer to execute event-based applications. It allows the user to merely focus on the application logic, while the effort of placing, distributing, and executing the application is handled by the framework. Several research efforts have used Storm to either evaluate new operator placement algorithms in a real environment or to propose some architectural improvements (e.g., [1, 14, 18, 24, 25]).

Apache Spark [27] is a general-purpose framework for large-scale processing, which provides a batch and micro-batch processing approach. This latter alternative is throughput-oriented, whereas Storm, which is a pure DSP system, can further minimize the application latency and thus can be preferred in latency sensitive scenarios. Another emerging framework is Apache Flink[1], which provides a unified solution for batch and stream processing.

Aside the specific functionalities, these frameworks use directed graphs to model DSP applications; therefore, our ODP formulation well represents their placement problem. Since several placement policies [1, 6, 7, 12, 19, 25] have been already integrated into Storm, we have chosen this framework to implement our ODP scheduler and compare its performance to that achieved by some centralized [25] and decentralized heuristics [19, 21].

3. SYSTEM MODEL AND PROBLEM STATEMENT

Devising an optimal application deployment strongly depends on the assumptions made about the domain it will be applied to. A suitable model grasps these relevant assumptions, taking into account the heterogeneity of represented entities, i.e., application requirements and resource capabilities. Therefore, before going into the details of our placement model, we provide a formal description of the involved entities. For the sake of clarity, in Table 1 we summarize the notation used throughout the paper.

3.1 DSP Model

From an operational perspective, a DSP application is made of a network of operators connected by streams. An operator is a self-contained processing element that can execute a generic user-defined code, whether it is a predefined 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 application 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 with no incoming links) and sinks (i.e., nodes with no outgoing link), and the links in E_{dsp} represent the streams, i.e., data flows, between nodes. Due to the difficulties of formalizing a generic relationship between the operator code and its non-functional attributes, we consider each operator as a black-box component, assuming that its attributes, if not known a-priori, can be obtained thanks to empirical measurements or benchmarks. Each node $i \in V_{dsp}$ has the following attributes: C_i, the amount of resources required for its execution; and R_i, its execution time per unit of data on a reference processor. We assume, without loss of generality, that C_i is a scalar value, but our placement model can be easily extended to consider C_i as a vector of required resources. Furthermore, to avoid binding together applications and resources, we require R_i to be measured on a reference processor.

[1]https://flink.apache.org/

Table 1: Main notation adopted in the paper.

Symbol	Description
G_{dsp}	Graph representing a DSP application
V_{dsp}	Set of vertices (operators) of G_{dsp}
E_{dsp}	Set of edges (streams) of G_{dsp}
C_i	Resources required to execute $i \in V_{dsp}$
R_i	Execution time per data unit of $i \in V_{dsp}$ on a reference processor
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}
C_u	Amount of resources available on $u \in V_{res}$
S_u	Processing speed-up of $u \in V_{res}$
A_u	Availability of node $u \in V_{res}$
$d_{(u,v)}$	Network delay on $(u,v) \in E_{res}$
$A_{(u,v)}$	Availability of $(u,v) \in E_{res}$
$V^i_{res} \subseteq V_{res}$	Subset of nodes where $i \in V_{dsp}$ can be placed
$x_{i,u}$	Placement of $i \in V_{dsp}$ on $u \in V^i_{res}$
$y_{(i,j),(u,v)}$	Placement of $(i,j) \in E_{dsp}$ on $(u,v) \in E_{res}$

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 link across the networks which results by the underlying physical network paths and routing strategies. The label associated with each node $u \in V_{res}$ specifies the following QoS attributes: C_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. The label associated with each link $(u,v) \in E_{res}$, with $u,v \in V_{res}$, specifies: $d_{(u,v)}$, the network delay between node u and v; and $A_{(u,v)}$, the link availability, i.e., the probability that the link between u and v is active. This model considers also loop links, i.e., edge 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., Software Defined Networking — SDN).

3.3 Operator Placement Problem

The DSP placement problem consists in determining a suitable mapping between the DSP graph G_{dsp} and the resource graph G_{res} in a such a way that all constraints are fulfilled. Figure 1 represents a simple instance of the prob-

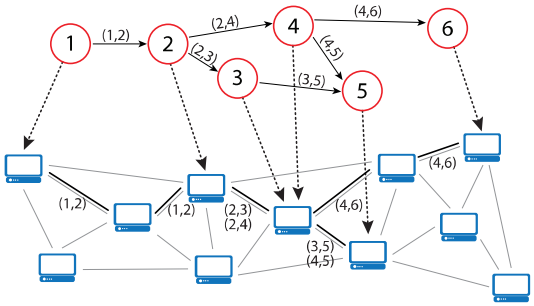


Figure 1: Placement of the application operators on the computing and network resources

lem. 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, political). 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 $\forall i \in I, |V^i_{res}| = 1$.

We can conveniently model the DSP placement with binary variables $x_{i,u}$, $i \in V_{dsp}$, $u \in V^i_{res}$: $x_{i,u} = 1$ if operator i is deployed on node u and $x_{i,u} = 0$ otherwise. A correct placement must deploy an operator on one and only one computing node; this condition can be guaranteed requiring that $\sum_u x_{i,u} = 1$, with $u \in V^i_{res}$, $i \in V_{dsp}$. We also find convenient to consider binary variables associated to links, namely $y_{(i,j),(u,v)}$, $(i,j) \in E_{dsp}$, $(u,v) \in E_{res}$, which denotes whether the data stream flowing from operator i to operator j traverses the network path from node u to node v. By definition, we have $y_{(i,j),(u,v)} = x_{i,u} \wedge x_{j,v} = x_{i,u} \cdot x_{j,v}$. For short, in the following we denote by $\boldsymbol{x}$ and $\boldsymbol{y}$ the placement vectors for nodes and edges, respectively, where $\boldsymbol{x} = \langle x_{i,u} \rangle$, $\forall i \in V_{dsp}$, $\forall u \in V^i_{res}$ and $\boldsymbol{y} = \langle y_{(i,j),(u,v)} \rangle$, $\forall x_{i,u}, x_{j,v} \in \boldsymbol{x}$.

4. OPTIMAL PLACEMENT MODEL

There are several strategies to determine the deployment of a DSP application on a set of computing resources, as reviewed in Section 2. Each strategy has been developed following a specific objective, which, ultimately, can be generalized in the optimization of a specific utility function, such as the the exchanged traffic, user-perceived latency, and resource utilization. In this section, exploiting tools provided by the optimization theory, we propose a model for the optimal operator placement problem that can be adjusted to satisfy different utility functions. First, we present a simple version of the model, which allows to better argument the design choices. Then, in Section 5, we illustrate the challenges of modeling other QoS constraints and attributes.

4.1 QoS Metrics

We first consider the application response time and availability, which are primarily user-oriented rather than system-oriented QoS metrics.

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:

$$R = \max_{p \in \pi_{G_{dsp}}} R_p \tag{1}$$

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:

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

where

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

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

denote respectively the execution time of operator i when mapped on node u and the network delay for transferring data from i to j when mapped on the path from u to v, where $i, j \in V_{dsp}$ and $u, v \in V_{res}$.

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 possibly complex dependency relationship among logical components sharing physical nodes and networks links. This will be subject of future work.

With the independence assumption, we readily have:

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

where

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

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

denote respectively the availability of the operator $i \in V_{dsp}$ and of the data stream from i to j, $(i,j) \in E_{dsp}$. To avoid dealing with multiplication, we consider the logarithm of the availability, obtaining:

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

where $a_u = \log A_u$ and $a_{(u,v)} = \log A_{(u,v)}$.
Expression (8) deserves some comments. Let us focus on the first term and observe that the logarithm of the first factor of $A(\boldsymbol{x}, \boldsymbol{y})$, that is $\prod_{i \in V_{dsp}} A_i(\boldsymbol{x}) = \prod_{i \in V_{dsp}} (\sum_{u \in V^i_{res}} A_u x_{i,u})$,

is actually $\sum_{i \in V_{dsp}} \log(\sum_{u \in V^i_{res}} A_u x_{i,u})$ and in general $\log(\sum_{u \in V^i_{res}} A_u x_{i,u}) \neq \sum_{u \in V^i_{res}} (\log A_u) x_{i,u}$. However, since only one term of the sum in the expression $\log(\sum_{u \in V^i_{res}} A_u x_{i,u})$ can be different from zero (an operator is assigned to exactly one node which implies that only variable in the set $\{x_{i,u}\}_{u \in V^i_{res}}$ is equal to 1) it follows that for any application placement $\boldsymbol{x}$, $\log(\sum_{u \in V^i_{res}} A_u x_{i,u}) = \sum_{u \in V^i_{res}} (\log A_u) x_{i,u}$, from which the first term in (8) directly follows. Similar arguments apply to the second term.

4.2 Optimal Placement Formulation

In a feasible assignment $\boldsymbol{x}$ (and associated $\boldsymbol{y}$), the computing resources $u \in V_{res}$ execute the application operators $i \in V_{dsp}$ with respect to their capabilities (i.e., C_u). Nevertheless, not all feasible assignments produce desirable application performances, thus we introduce a utility function that analytically defines an order relationship among all feasible solutions. The optimal placement results from the maximization of a given utility function. Depending on the utilization scenario, the utility function could be aimed at optimizing specific QoS attributes. These different optimization goals could be possibly conflicting, thus leading to a multi-objective optimization problem, which can be transformed into a single objective problem, using the Simple Additive Weighting (SAW) technique [26]. According to SAW, we define the utility function $F(\boldsymbol{x}, \boldsymbol{y})$ as a weighted sum of the normalized QoS attributes of the application, as follows:

$$F(\boldsymbol{x}, \boldsymbol{y}) = w_r \frac{R_{\max} - R(\boldsymbol{x}, \boldsymbol{y})}{R_{\max} - R_{\min}} + w_a \frac{\log A(\boldsymbol{x}, \boldsymbol{y}) - \log A_{\min}}{\log A_{\max} - \log A_{\min}} \quad (9)$$

where $w_r,\ w_a \geq 0$, $w_r + w_a = 1$, are weights for the different QoS attributes. $R_{\max}$ ($R_{\min}$) and $A_{\max}$ ($A_{\min}$) denote, respectively, the maximum (minimum) value for the overall expected response time and availability. Observe that after normalization, each metric ranges in the interval $[0, 1]$, where the value 1 corresponding to the best possible case, that is $R(\boldsymbol{x}, \boldsymbol{y}) = R_{\min}$, $\log A(\boldsymbol{x}, \boldsymbol{y}) = \log A_{\max}$, and 0 to the worst case, $R(\boldsymbol{x}, \boldsymbol{y}) = R_{\max}$, $\log A(\boldsymbol{x}, \boldsymbol{y}) = \log A_{\min}$.

The Optimal DSP Placement (ODP) can be formulated as an Integer Linear Programming (ILP) model as follows:

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

where

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

subject to:

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

$$\sum_{i \in V_{dsp}} C_i x_{i,u} \leq C_u \quad \forall u \in V_{res} \quad (11)$$

$$\sum_{u \in V^i_{res}} x_{i,u} = 1 \quad \forall i \in V_{dsp} \quad (12)$$

$$x_{i,u} = \sum_{v \in V^j_{res}} y_{(i,j),(u,v)} \ \forall (i,j) \in E_{dsp},\ u \in V^i_{res} \quad (13)$$

$$x_{j,v} = \sum_{u \in V^i_{res}} y_{(i,j),(u,v)} \ \forall (i,j) \in E_{dsp}, v \in V^j_{res} \quad (14)$$

$$x_{i,u} \in \{0,1\} \quad \forall i \in V_{dsp}, u \in V^i_{res} \quad (15)$$

$$y_{(i,j),(u,v)} \in \{0,1\} \ \forall (i,j) \in E_{dsp}, (u,v) \in V^i_{res} \times V^j_{res} \quad (16)$$

In the problem formulation we replaced the utility function $F(\boldsymbol{x}, \boldsymbol{y})$ with the objective function $F'(\boldsymbol{x}, \boldsymbol{y}, r)$ which is obtained from $F(\boldsymbol{x}, \boldsymbol{y})$ 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 not linear 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 not linear term, F' is linear in r as well as in $\boldsymbol{x}$ and $\boldsymbol{y}$.

Equation (10) follows from (1)–(4). 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 (11) limits the placement of operators on a node $u \in V_{res}$ according to its available resources; Equation (12) guarantees that each operator $i \in V_{dsp}$ is placed on one and only one node $u \in V^i_{res}$. Finally, constraints (13)–(14) model the logical `AND` between the placement variables, that is, $y_{(i,j),(u,v)} = x_{i,u} \wedge x_{j,v}$. It is worth observing that the proposed `AND` constraints formulation is motivated by recent work in [11] which, albeit in a different context, provides empirical evidence that this specific `AND` formulation leads to reduced computational time with respect to alternative formulations[2]. For the sake of comparison, we compared this formulation to the alternatives (also taken from [11]) and, interestingly, we experienced - on average - a tenfold speed-up.

THEOREM 1. *The Optimal DSP Placement problem is an NP-hard problem.*

PROOF. In order to verify the NP-hardness of the Optimal DSP Placement problem, it suffices to prove that the corresponding decision problem is NP-hard. The decision problem can be formulated as follows: For a given DSP application and a set of computing and network resources, is there a feasible placement of the application? To prove the NP-hardness of this decision problem, let us consider the special case where: the resources required by the DSP application C_{i_k} are integers, where $k = \{1, \ldots, n\}$ and n is the number of operators; the network has only two nodes, $V_{res} = \{u, v\}$, with capacity $C_u = C_v = (\sum_{k=1}^{n} C_{i_k})/2$; there are no network delays, that is $d_{u,v} = d_{v,u} = 0$, which allows us to ignore the variables $\boldsymbol{y}$; and operators can be placed on both nodes, $V^{i_k}_{res} = V_{res}$, $k = \{1, \ldots, n\}$. It is easy to realize that the resulting problem is the well known Partition problem [13] which is known to be NP-hard. Since this special case is NP-hard, the general decision problem is NP-hard as well. And since the original optimization problem is at least as hard as the decision problem, it follows that Optimal DSP Placement problem is NP-hard as well. □

[2]In [11], the authors provide experimental evidence that the constraints (13)–(14) yields *better* lower bounds with respect to alternative constraints formulations. This results in more aggressive bounding in the *bound* phase of the *branch and bound* algorithm of ILP solvers and thus less iterations and faster convergence to the optimal solution.

5. NETWORK-RELATED EXTENSION AND QOS METRICS

The ODP model can be easily extended to account for other constraints and user- and/or system-oriented QoS metrics, such as energy consumption, bandwidth constraints, monetary cost, privacy. In this section, without lack of generality, we include network-related QoS metrics that have been widely used in the literature to assess the quality of the DSP placement algorithms, such as network usage [21], inter-node traffic [1, 25], and the so called elastic energy [19].

Bandwidth constraint. In the ODP formulation, we made the implicit assumption that the data streams traffic between operators does not saturate the logical link bandwidth. Since this assumptions might not hold true in practice, we can explicitly account for the limited logical link capacity in ODP as follows. For each pair of communicating operators i and j, $(i,j) \in E_{dsp}$, let $\lambda_{(i,j)}$ denote the average data traffic rate and, for each logical link $(u,v) \in E_{res}$, let $B_{(u,v)}$ denote available bandwidth. Then, similarly to the node computing resource constraints (11), we have the following network link available bandwidth constraints:

$$\sum_{(i,j)\in E_{dsp}} \lambda_{(i,j)} y_{(i,j),(u,v)} \leq \quad B_{(u,v)} \; \forall u \in V_{res}, v \in V_{res} \quad (17)$$

It is important to observe that, while the amount of available node computing capacity C_u in (11) is a known quantity, the amount of available bandwidth $B_{(u,v)}$ in (17) needs to be estimated since it depends on the (typically unknown) network traffic which traverses the physical links comprising the network path between node u and v[3]. Unfortunately, bandwidth estimation is known to be a non straightforward process as it requires active - and rather traffic intensive - end-to-end measurements techniques [20]. The situation, nevertheless, dramatically changes if we consider networks with advanced capabilities, e.g., SDN. In these networks, it becomes possible to have access to network information and/or allocate resources so as to reserve the desired amount of bandwidth on each logical link [8].

Network-related QoS metrics. Following the network-aware DSP placement policies in the literature [1, 19, 21, 25], we define the following metrics: the inter-node traffic T, the network usage N, and an approximation of the elastic energy EE. Let $Z(\boldsymbol{y})$, $Z = T|N|EE$, denote the QoS attribute of the DSP application under the DSP placement policy $\boldsymbol{y}$, we have:

$$Z(\boldsymbol{y}) = \sum_{(i,j)\in E_{dsp}} Z_{(i,j)}(\boldsymbol{y}) \quad (18)$$

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:

$$T_{(i,j)}(\boldsymbol{y}) = \sum_{(u,v)\in V^i_{res}\times V^j_{res}: u\neq v} \lambda_{(i,j)} y_{(i,j),(u,v)} \quad (19)$$

where $\lambda_{(i,j)}$ is the data rate of the stream.

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:

$$N_{(i,j)}(\boldsymbol{y}) = \sum_{(u,v)\in V^i_{res}\times V^j_{res}: u\neq v} \lambda_{(i,j)} d_{(u,v)} y_{(i,j),(u,v)} \quad (20)$$

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

In their paper [19], 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 (see Section 2). 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:

$$EE_{(i,j)}(\boldsymbol{y}) = \sum_{(u,v)\in V^i_{res}\times V^j_{res}: u\neq v} \lambda_{(i,j)} d^2_{(u,v)} y_{(i,j),(u,v)} \quad (21)$$

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

Network-related utility function. The utility function $F(\boldsymbol{x},\boldsymbol{y})$, previously defined in Equation (9), can be readily re-written to include the network-related QoS metric as follows:

$$F_{net}(\boldsymbol{x},\boldsymbol{y}) = F(\boldsymbol{x},\boldsymbol{y}) + w_z \frac{Z_{\max} - Z(\boldsymbol{y})}{Z_{\max} - Z_{\min}} \quad (22)$$

where $w_z \geq 0$ weighs the network-related attribute, $w_r + w_a + w_z = 1$, $Z = T|N|EE$, and $Z_{\max}$ and $Z_{\min}$ denote respectively the maximum and the minimum value for the network term.

For sake of clarity, we report the new formulation of the placement problem that considers the bandwidth constrains and the network-related metrics:

$$\max_{\boldsymbol{x},\boldsymbol{y},r} F'_{net}(\boldsymbol{x},\boldsymbol{y},r)$$

where

$$F'_{net}(\boldsymbol{x},\boldsymbol{y},r) = F'(\boldsymbol{x},\boldsymbol{y},r) + w_z \frac{Z_{\max} - Z(\boldsymbol{y})}{Z_{\max} - Z_{\min}}$$

subject to:

$$\begin{aligned}
r \geq & \sum_{k=1}^{n_p} \sum_{u\in V^{i_k}_{res}} \frac{R_{i_k}}{S_u} x_{i_k,u} + \\
& \sum_{k=1}^{n_p-1} \sum_{(u,v)\in V^{i_k}_{res}\times V^{i_{k+1}}_{res}} d_{(u,v)} y_{(i_k,i_{k+1}),(u,v)} \quad \forall p \in \pi_G \\
B_{(u,v)} \geq & \sum_{(i,j)\in E_{dsp}} \lambda_{(i,j)} y_{(i,j),(u,v)} \quad \forall u \in V_{res}, v \in V_{res}
\end{aligned} \quad (23)$$

$$\sum_{i\in V_{dsp}} C_i x_{i,u} \leq C_u \quad \forall u \in V_{res}$$

$$\sum_{u\in V^i_{res}} x_{i,u} = 1 \quad \forall i \in V_{dsp}$$

$$x_{i,u} = \sum_{v\in V^j_{res}} y_{(i,j),(u,v)} \quad \forall (i,j) \in E_{dsp},\ u \in V^i_{res}$$

$$x_{j,v} = \sum_{u\in V^i_{res}} y_{(i,j),(u,v)} \quad \forall (i,j) \in E_{dsp},\ v \in V^j_{res}$$

[3] Actually, since two different logical links could share some physical links, the constraints (17) should be expressed in terms of the physical and not logical links capacity. This is, in general, not feasible since it would require complete knowledge of: 1) the physical network topology, 2) the links characteristics, and 3) the routing tables.

$$x_{i,u} \in \{0,1\} \quad \forall i \in V_{dsp}, u \in V_{res}^i$$
$$y_{(i,j),(u,v)} \in \{0,1\} \quad \forall (i,j) \in E_{dsp}, (u,v) \in V_{res}^i \times V_{res}^j$$

where Equation (23) limits the amount of data that flows on the logical link $(u,v) \in E_{res}$, according to its available bandwidth $B_{(u,v)}$.

6. STORM INTEGRATION

To enable the usage of ODP in a real DSP framework, we have developed a prototype scheduler for Apache Storm, named S-ODP. We first briefly describe the main features of Storm and how it represents and executes DSP applications. Then, we present the prototype design in details.

6.1 Apache Storm

Storm is an open source, real-time, and scalable DSP system maintained by the Apache Software Foundation. It provides an abstraction layer where event-based applications can be executed over a set of worker nodes interconnected by an overlay network. A *worker node* is a generic computing resource (e.g., a physical host, a mobile device, a virtual machine, a Docker container), whereas the overlay network comprises the logical links among these nodes.

In Storm, an application is represented by its *topology*, which is a DAG with spouts and bolts as vertices and streams as edges. A *spout* is a data source that feeds the data into the system through one or more streams. A *bolt* is either a processing element, which extracts valuable information from incoming tuples, or a final information consumer; a bolt can also generate new outgoing streams, like spouts do. A *stream* is an unbounded sequence of *tuples*, which are key-value pairs. We refer to spouts and bolts as operators. Figure 2a shows an example of a DSP application.

Storm uses three types of entities with different grain to execute a topology: tasks, executors, and worker processes. A *task* is an instance of an application operator (i.e., spout or bolt), and it is in charge of a share of the incoming operator stream. An *executor*, which is the smallest schedulable unit, can execute one or more tasks related to the same operator. A *worker process* is a Java process that runs a subset of the executors of the *same* topology, i.e., a topology can be distributed across different worker processes. As represented in Figure 2b, there is an evident hierarchy among the Storm entities: a group of tasks runs sequentially in the executor, which is a thread within the worker process, that in its turn serves as container on the worker node.

Besides the computing resources, i.e., the worker nodes, the Storm architecture includes two components: Nimbus and ZooKeeper. *Nimbus* is a centralized component in charge of coordinating the topology execution; it uses its *scheduler* to define the placement of the application operators on the pool of available worker nodes. The assignment plan determined by the scheduler is communicated to all the worker nodes through *ZooKeeper*, which is a shared in-memory service for managing configuration information and enabling distributed coordination[4]. Since each worker node can execute one or more worker processes, a *Supervisor* component on each worker node starts or terminates worker processes on the basis of the Nimbus assignments. Each worker node can concurrently run a limited number of worker processes, based on the number of available *worker slots*.

[4]http://zookeeper.apache.org/

6.2 S-ODP: ODP in Storm

We develop a new scheduler for Storm, named **S-ODP**, whose core is the model presented in Section 5, where we consider the data rate exchanged among the application operators. In order to design S-ODP, we have to address two issues: (1) how to adapt the S-ODP formulation to the specific execution entities of Storm, and (2) how to instantiate the ODP model with the proper QoS information about computing and networking resources.

As regards the first issue, we have to model the fact that the Storm scheduler places the application executors on the available worker slots, considering that at most $EPS_{\max}$ executors can be co-located on the same slot. Hence, S-ODP defines $G_{dsp} = (V_{dsp}, E_{dsp})$, with V_{dsp} as the set of executors and E_{dsp} as the set of streams exchanged between the executors. Since in Storm an executor is considered as a black box element, we conveniently assume unitary its attributes, i.e., $C_i = 1$ and $R_i = 1$, $\forall i \in V_{dsp}$. The resource model $G_{res} = (V_{res}, E_{res})$ must take into account that a worker node $u \in V_{res}$ offers some worker slots $WS(u)$, and each worker slot can host at most $EPS_{\max}$ executors. For simplicity, S-ODP considers the amount of available resources C_u on a worker node $u \in V_{res}$ equals to the maximum number of executors it can host, i.e., $C_u = WS(u) \times EPS_{\max}$.

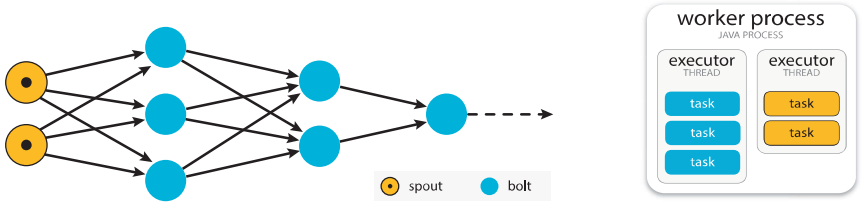
As regards the second issue, Storm allows to easily develop new centralized schedulers with the pluggable scheduler APIs. However, Storm is not aware of the QoS attributes of its networking and computing resources, except for the number of available worker slots. Since we need to know these QoS attributes to apply the ODP model, we rely on the Storm extension[5] we presented in [6], that enables the QoS awareness of the scheduling system by providing intra-node (i.e., availability) and inter-node (i.e., network delay and exchanged data rate) information. This extension estimates network latencies using a network coordinate system, which is built through the Vivaldi algorithm [9], a decentralized algorithm having linear complexity with respect to the number of network locations. S-ODP retrieves, from the monitoring components of the extended Storm, the information needed to parametrize the nodes and edges in G_{dsp} and G_{res}. Specifically, it considers: the average data rate exchanged between communicating executors (i.e., $\lambda_{(i,j)}, \forall (i,j) \in E_{dsp}$), the node availability ($A_u, u \in V_{res}$), and the network latencies ($d_{(u,v)}, \forall u,v \in V_{res}$). Once built the ODP model, S-ODP relies on CPLEX© (version 12.6.2) for solving the placement problem.

From an operative prospective, Nimbus uses S-ODP to compute the optimal placement when a new application is submitted to Storm and when a failure of the worker process compromises the application execution. In the latter case, S-ODP invalidates the existing assignment and computes the new optimal placement. When information on the exchanged data rate is unknown (e.g., first run), S-ODP defines an early assignment and monitors the application execution to harvest the needed information. Then, S-ODP reassigns the application, solving the updated ODP model with the network-related QoS attributes.

7. EXPERIMENTAL RESULTS

We analyze an extensive set of experimental results that aim at demonstrating the flexibility of the ODP formulation

[5]Source code available at http://bit.ly/extstorm



(a) Topology of a DSP application (b) Execution model

Figure 2: Storm abstractions

and investigating its scalability. Specifically, using S-ODP, we first analyze in Section 7.1 how the ODP formulation allows us to consider the optimization of user-oriented QoS metrics, such as response time and availability. Then, in Section 7.2 using S-ODP we demonstrate how our formulation represents a general framework and a benchmark for DSP placement optimization. To this end, we compare the performance achieved by some centralized and decentralized placement heuristics to that obtained by their corresponding optimal formulation based on ODP. Finally, since ODP is formulated as an ILP problem, we investigate through numerical experiments its scalability, analyzing the relationship between the solver resolution time and some model parameters, such as the application size and the number of resources. We close this section evaluating two simple approaches that can reduce the solver resolution time with a trade-off on the quality of the computed solution.

7.1 Optimizing User-oriented QoS Metrics

The ODP model we presented in Section 4 allows us to define the placement by optimizing different QoS attributes, whose importance depends on the utilization scenario. In this experiment, we use the S-ODP prototype to optimize different user-oriented QoS metrics, namely application response time and availability.

We perform the experiments using Apache Storm 0.9.3 on a cluster of 8 worker nodes, each with 2 worker slots, and 2 further nodes for Nimbus and ZooKeeper. A worker slot can host at most 4 executors, i.e., $EPS_{\max} = 4$. Each node is a virtual machine with one vCPU and 2 GB of RAM. We emulate wide-area network latencies among the Storm nodes using *netem*, which applies to outgoing packets a Gaussian delay with mean and standard deviation in the ranges $[12, 32]$ ms and $[1, 3]$ ms, respectively. Furthermore, half of the worker nodes has an availability of 99 % and the other of 100 %, whereas the links are always available. As test-case application we use Tag-and-Count, which tags and counts the sentences produced by a data source. Its topology is represented in Figure 3 and is composed by a source, which generates 10 tuples/s, followed by a sequence of 5 operators before reaching the final consumer. The source and the consumer are pinned operators, that are instantiated with a single executor each; all the others are instantiated with two executors each.

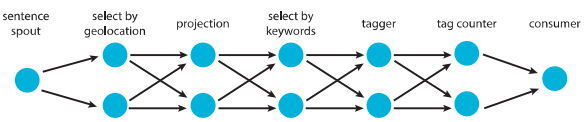


Figure 3: Tag-and-Count application

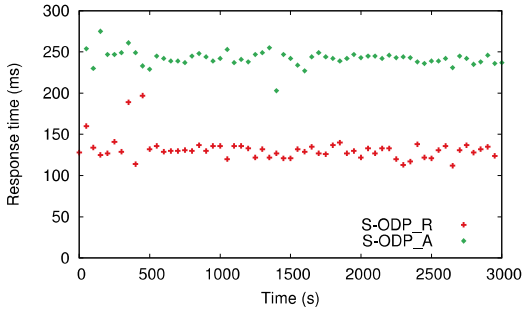
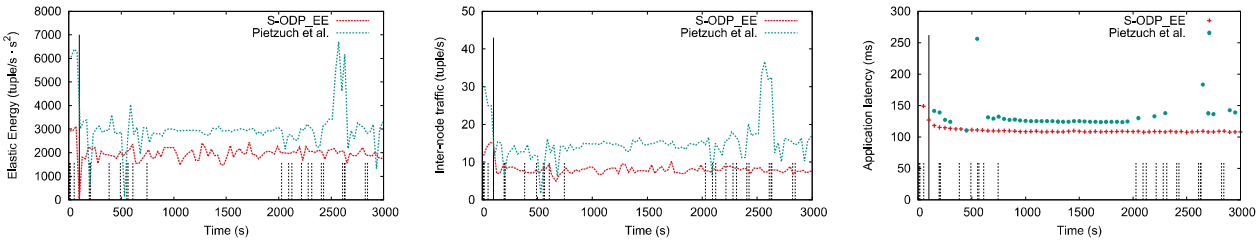


Figure 4: Application performance when ODP optimizes different user-oriented QoS metrics: response time (S-ODP_R) and availability (S-ODP_A)

We evaluate the effects on the application performance of two different configurations of S-ODP, namely S-ODP_R and S-ODP_A. **S-ODP_R** computes the placement by optimizing, as QoS metric, the response time R (we recall that R is the worst end-to-end delay on the source-sink path), i.e., it solves the ODP model with the utility function parametrized with weights $w_r = 1$ and $w_a = w_z = 0$. **S-ODP_A** maximizes the application availability A, by solving ODP with weights $w_a = 1$ and $w_r = w_z = 0$. Figure 4 shows the effects of the two utility functions on the application performance, expressed in terms of response time R. S-ODP_A places all the executors on the most available nodes, therefore the resulting application availability is 100%. However, since the scheduler does not consider the network latencies, the application experiences a response time that is, on average, 1.8 times higher than that achieved with S-ODP_R. The latter obtains the lowest response time, but, on the other hand, places 8 of 12 executors on worker nodes with 99% of availability. The resulting application availability is 92.27%, which is perceived with an increment of the tuple loss rate (3.37% during the experiment).

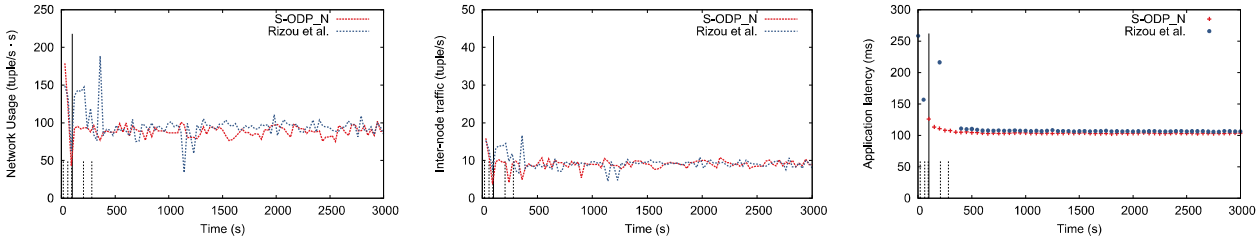
7.2 Evaluating Placement Heuristics

In this set of experiments we use the S-ODP prototype to evaluate some placement heuristics proposed in literature, namely the Pietzuch's, Rizou's, and Xu's algorithms (i.e., [19, 21, 25]). Since each solution has its own optimization goal, we conveniently adapt the ODP model as presented in Section 5. We use the same execution environment and application described in Section 7.1, except for 100% availability of all the nodes and links. Furthermore, since network links have high available bandwidth, S-ODP omits the bandwidth constraints expressed in Equation (17). This setting allows for a fair comparison with the



(a) Elastic energy (b) Inter-node traffic (c) Application latency

Figure 5: Comparison of placement decisions: Pietzuch's solution and optimal assignment (S-ODP_EE)



(a) Network usage (b) Inter-node traffic (c) Application latency

Figure 6: Comparison of placement decisions: Rizou's solution and optimal assignment (S-ODP_N)

other placement solutions, here investigated, which do not take these QoS attributes into account.

Since the objective function of S-ODP needs the exchanged data rate information known only at run-time, S-ODP computes the placement in two steps, as described in Section 6.2. First, the scheduler defines the placement using weights $w_r = w_z = 0$ and $w_a = 1$. Then, it solves again the ODP model with weights $w_r = w_a = 0$ and $w_z = 1$, and reassigns the application. This rescheduling event happens at 100 s, and is represented with a vertical full line in the following figures. Also the schedulers under investigation can perform run-time reassignments; these events are represented with vertical dot-dash lines.

In this first experiment, we compare the performance of the decentralized Pietzuch's algorithm [19] we implemented in [6] with respect to the optimal placement computed by **S-ODP_EE**. The latter is the centralized S-ODP scheduler set up so to minimize the elastic energy of the spring system that represents the application, see Equation (21). Figure 5 reports the optimized metric (i.e., elastic energy), the average inter-node traffic, and the average application latency (i.e., the end-to-end delay of the tuples). We compute the elastic energy (Figure 5a) relying on real network latencies, because this allows to evaluate the behaviour of the Pietzuch's solution, neglecting the noise introduced by the network latency estimation system. The Pietzuch's algorithm finds a sub-optimal solution: considering the interval from 800 s to 2000 s, the inter-node traffic is on average 1.84 times higher than that achieved with the optimal placement, and the application latency is 1.15 times higher. The fluctuations introduced by the network latency estimation system are detrimental for this scheduling solution, which is very susceptible to this metric. As a consequence, even if no real environmental or application changes occur, the Pietzuch's algorithm can trigger some reassignments; see, for example, the behaviour of the system after 2000 s. The large number of reassignments required to reach a stable configuration (16 in this experiment) and the related stop-and-replay of the involved operators can negatively impact the application. Indeed, during the transient periods the application experiences unavailability, tuple loss, and peaks in traffic (Figure 5b) and latency (Figure 5c).

With the second experiment, we evaluate the decentralized Rizou's algorithm [21] with respect to the optimal placement determined by **S-ODP_N**, that is S-ODP configured to minimize the network usage, see Equation (20). Rizou et al. evaluated their proposal only through simulation. We have implemented their algorithm in Storm in order to effectively compare its performance against the optimal placement achieved by S-ODP_N. Figure 6 reports the optimized metric (i.e., network usage), the inter-node traffic, and the overall application latency. Differently from the Pietzuch's solution, the Rizou's one directly minimizes the network usage. Comparing the Rizou's scheduling policy with the optimal one, we make the following observations. First, Rizou's shows good convergence, better than the Pietzuch's solution: a stable assignment is found in less than 300 s, after only 5 reassignments, thus confirming the simulation results presented by Rizou et al. in [21]. Second, latency weighs less on the scheduler's reassignment decisions, which makes the system less susceptible to fluctuations in its estimation. Lastly, in this specific, although simple, experimental scenario, the Rizou's algorithm finds the optimal placement for the application, effectively minimizing the network usage.

In the third experiment we evaluate the centralized scheduling policy proposed by Xu et al. [25] with respect to **S-ODP_T**, i.e., S-ODP that minimizes the inter-node traffic T, see Equation (19). In this case, no latency information is needed. Although the authors developed their solution for Storm, its code is not publicly available, so we re-implemented it adhering to the description in [25]. Differently from the previous heuristics, the Xu's scheduler is a centralized solution with a complete knowledge of the network, thus it can determine the placement relying

on global information. Similarly to S-ODP, this scheduler needs a preliminary run of the application in order to extract bandwidth-related information. Hence, it initially places the application using a round-robin strategy; then, as soon as data rate information is available, it reassigns the operators using a best-fit heuristic. The latter tries to co-locate operators in decreasing order of exchanged data rate. Figure 7 compares the inter-node traffic achieved by the Xu's scheduler and our S-ODP_T. The Xu's algorithm performs a first reassignment at 100 s, which produces a sub-optimal configuration. In particular, there is a sequence of 3 executors that run on two nodes and exchange data using two network links instead of one. However, the Xu's scheduler detects the problem and fixes the application placement at 750 s, achieving the optimal solution computed by S-ODP_T.

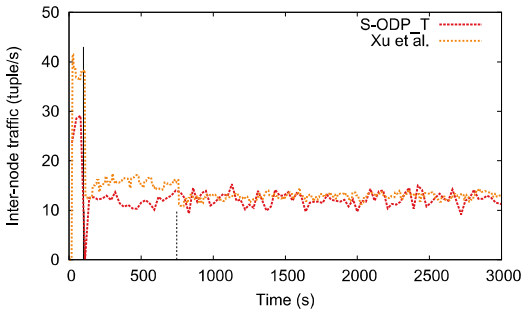


Figure 7: Comparison of placement decisions: Xu's solution and optimal assignment (S-ODP_T)

In all these experiments, S-ODP computes the optimal solution in less than 420 ms.

7.3 Scalability Analysis

We now focus on the scalability analysis of ODP. To evaluate its computational cost, we solve the ILP problem on randomly generated problem instances, using an Amazon EC2 virtual machine (c4.xlarge with 4 vCPU and 7.5 Gb RAM). As computational cost metric, we use the resolution time experienced by CPLEX. To avoid interferences among the model parameters, we use a baseline scenario and change a single factor at a time. The baseline scenario defines applications, computing and network resources with homogeneous characteristics. This represents the worst-case scenario for the CPLEX solver that, using a branch-and-cut resolution strategy, has to explore the whole solution space in order to find the optimum. If not otherwise specified, applications and resources are parametrized as reported in Table 2.

Table 2: Parameters of application and resource models.

Model	Parameter	Value
Application	size of V_{dsp}	20
	R_i	1.0 s
	C_i	1
Resource	size of V_{res}	20
	A_u	100%
	C_u	4
	S_u	1.0
	$d_{(u,v)}$	$\max\{x \in \text{Gaussian } \mathcal{N}(22,5), 1\}$ ms
	$A_{(u,v)}$	100%

We consider two alternatives of layered topology for G_{dsp}, representing *sequential* and *fat* DSP applications, where each

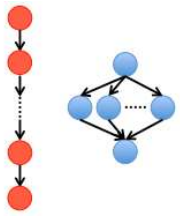


Figure 8: Sequential (left) and fat (right) applications

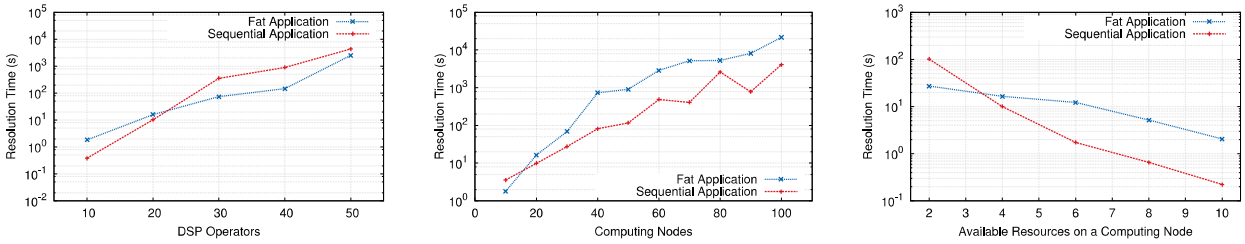
layer has one or more operators. The first and last layer contain the sources and sinks of the application, respectively. The DAGs of sequential (also known as pipelined) and fat applications are shown in Figure 8.

In the following experiments, G_{res} models a geographically distributed infrastructure, where computing nodes are interconnected with not-negligible network delays. We also assume that G_{res} is a fully connected graph, i.e., there always exists a logical link $(u, v) \in E_{res}$ between any two computing resources $u, v \in V_{res}$. We evaluate the ODP resolution time in relation to: *a)* the number of application operators, *b)* the number of computing resources, *c)* the percentage of available resources required by the application, and *d)* the capacity of each computing node.

In the first experiment, we consider the ODP resolution time when the number of operators in the application increases from 10 to 50. Figure 9a shows the results. Note that, with 50 operators, the placement vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ contain 1000 and 19600 or 38400 variables for the sequential and fat applications, respectively.

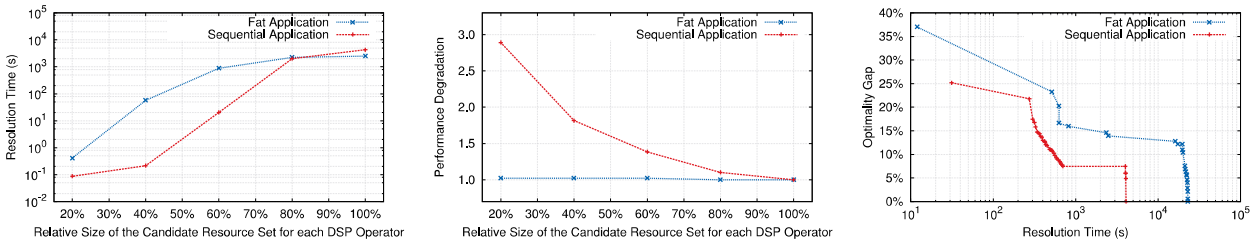
The second experiment, depicted in Figure 9b, evaluates the resolution time when the number of computing resources ranges from 10 to 100 and the number of application operators is fixed to 20. With 100 computing resources, the placement vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ contain 2000 and 190K or 360K variables for the sequential and fat applications, respectively. It is worth observing that the ODP resolution time is more sensible to the increment in the number of application operators rather than the number of computing resources. For example, consider the scenarios $Sc_1 = \{V_{res} = 20, V_{dsp} = 50\}$ and $Sc_2 = \{V_{res} = 50, V_{dsp} = 20\}$. Both of them generate a placement vector $\boldsymbol{x}$ with 1000 variables; however, in Sc_2 (where the number of computing resources is increased), the resolution time decreases from 2 to 36 times for sequential and fat applications respectively, although the placement vector $\boldsymbol{y}$ is 2.38 times larger. The complexity of CPLEX does not easily reveal the motivations behind this behaviour.

In the last experiment we investigate the impact of the amount of resources available on each computing node (C_u). We execute the applications with 20 operators on a network of 20 computing nodes, where the number of available resources C_u for each node $u \in V_{res}$ increases from 2 to 10. Figure 9c shows the results. The decreasing trend is readily motivated observing the objective function in Equation (9): since it penalizes network delays, the optimal placement is trivially determined when there is enough residual capacity on an already selected node, because operators are co-located on the same node. In an additional experiment not reported for the sake of space, we found that the load measured in terms of resources required by the application does not affect the resolution time.



(a) Number of application operators $|V_{dsp}|$ (b) Number of computing nodes $|V_{res}|$ (c) Number of available resources C_u on each computing node

Figure 9: Computational cost of solving the ODP model in different settings



(a) Resolution time using a randomly sampled set of resources (b) Performance degradation using a randomly sampled set of resources (c) Feasible solutions and advances on the optimality gap

Figure 10: Effects of simple heuristic approaches on the ODP resolution time

As expected, the scalability experiments show that, when the problem cardinality increases significantly, solving analytically the ODP model is not feasible, even if CPLEX is very efficient. Even if efficient heuristics are required to solve large scale problems, we can consider two simple strategies to reduce the model resolution time: *a)* sampling the set of candidate computing nodes, and *b)* fixing an upper bound on the resolution time.

The benefits and drawbacks of reducing through sampling the set of candidate computing resources V_{res}^i where each operator i can be placed, are illustrated in Figures 10a and 10b, respectively. We define as *performance degradation* (Figure 10b) the ratio between the value of the objective function when the placement occurs only on a subset of candidate resources and the optimal value when all the resources can be used. We use a simple random sampling to identify the candidate resource subset for each operator. We run the experiment with 20 computing nodes and 50 operators for both the types of application topology. We observe from Figure 10a that the resolution time decreases tremendously (i.e., contracting up to about 5 orders of magnitude), when each operator has a subset of candidates that is less than 40% of all the available resources. Figure 10b points out that the solution quality strictly depends on the application topology: with only 20% of computing nodes, the fat application experiences a performance degradation only of 1.02, whereas the resolution time of the sequential application is about 3 times higher than the optimal one.

When we define an upper bound for the resolution time, CPLEX tries to find the optimal solution until the time interval is not exceeded. Then, the solver returns the best known feasible and sub-optimal solution. An appropriate upper bound strictly depends on the problem instance, but in this experiment we want to show how CPLEX closes the optimality gap during the resolution time. The *optimality gap* is a metric used by CPLEX to represent the relative gap between the objective function of the best integer solution and the best feasible solution that results from the linearization of the model (branch-and-cut strategy). We use the configuration that reported the longest resolution time in the previous experiments: 100 computing nodes and 20 application operators. Figure 10c shows the trend of the optimality gap. Depending on the application topology, the solver behaves differently. For the sequential application, a near-optimal solution (i.e., with an optimality gap lower than 10%) is found in the first 500 s of computation. For the fat application, this convergence takes place just before the execution conclusion, thus it is harder to find a suitable upper bound for the resolution time. On the other hand, as shown in Figure 10b, the sampling technique seems to be a more effective strategy for the fat application.

8. CONCLUSIONS

In this paper we have presented a general formulation of the optimal placement problem (ODP) for data stream processing applications, which takes into account the heterogeneity of computing and networking resources. ODP can optimize different QoS requirements of the applications and we have considered the end-to-end latency, availability, and exchanged data-rate among operators. The optimal placement provided by the OPD solution can be used as a benchmark framework against which to compare other centralized and decentralized placement algorithms. To this end, we have developed an ODP-based prototype scheduler for Storm and have compared some well-known placement solutions proposed in the literature. As the placement problem is NP-hard, a heuristic approach is needed to solve it in a feasible amount of time. However, the extensive scalability analysis of ODP has shown that, with particular application topologies, simple strategies can be successfully used.

As future work, we plan to develop efficient heuristics to deal with large problem instances for the initial operator placement and to support online reconfigurations, where the placement decisions need to adapt at run-time in order to properly handle input streams or execution environments with continuously changing properties.

Our work opens several research directions, which include the modeling of other QoS attributes (e.g., elasticity, pay-as-you-go pricing, privacy) and of the cost of reconfiguration decisions (such as state transfer for stateful operators), and the adaptation of the placement model for other types of distributed applications (e.g., service composition, MapReduce). As long term research direction, we will consider the more general setting where a distributed Cloud infrastructure hosts multiple concurrent DSP applications, each arriving and departing over time with unforeseen requirements and characteristics. We will study the trade off between the QoS requirements of multiple DSP applications sharing the same infrastructure and the service provider objectives, e.g., profit maximization and/or optimal resource utilization.

9. ACKNOWLEDGMENTS

We thank Prof. Andrea Pacifici for his insightful comments and for suggesting us the elegant NP-hardness proof. This publication is supported by COST Action ACROSS (IC1304).

10. 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] N. Backman, R. Fonseca, and U. Çetintemel. Managing parallelism for stream processing in the cloud. In *Proc. of HotCDP '12*. ACM, 2012.

[3] A. Benoit, H. Casanova, V. Rehn-Sonigo, and Y. Robert. Resource allocation for multiple concurrent in-network stream-processing applications. *Parallel Computing*, 37(8):331–348, 2011.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the Internet of Things. In *Proc. of MCC '12*, pages 13–16. ACM, 2012.

[5] A. Brook. Low-latency distributed applications in finance. *ACM Commun.*, 58(7):42–50, 2015.

[6] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli. Distributed QoS-aware scheduling in Storm. In *Proc. of ACM DEBS '15*, pages 344–347, 2015.

[7] A. Chatzistergiou and S. D. Viglas. Fast heuristics for near-optimal task allocation in data stream processing over clusters. In *Proc. of ACM CIKM '14*, 2014.

[8] L. Cui, F. R. Yu, and Q. Yan. When big data meets software-defined networking: SDN for big data and big data for SDN. *IEEE Network*, 30(1):58–65, 2016.

[9] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. *SIGCOMM Comput. Commun. Rev.*, 34(4), 2004.

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

[11] G. Falcone, G. Nicosia, and A. Pacifici. Minimizing part transfer costs in flexible manufacturing systems: A computational study on different lower bounds. In *Proc. of UKSim '13*, pages 359–364, 2013.

[12] L. Fischer, T. Scharrenbach, and A. Bernstein. Scalable linked data stream processing via network-aware workload scheduling. In *Proc. of SSWS '13*, 2013.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman, 1979.

[14] T. Heinze, L. Aniello, L. Querzoni, and Z. Jerzak. Cloud-based data stream processing. In *Proc. of ACM DEBS '14*, pages 238–245, 2014.

[15] J. Huang, G. Liu, Q. Duan, and Y. Yan. QoS-aware service composition for converged network-cloud service provisioning. In *Proc. of IEEE SCC '14*, 2014.

[16] Y. Huang, Z. Luan, R. He, and D. Qian. Operator placement with QoS constraints for distributed stream processing. In *Proc. of 7th Int'l Conf. on Network and Service Management*, CNSM '11, 2011.

[17] G. T. Lakshmanan, Y. Li, and R. Strom. Placement strategies for Internet-scale data stream systems. *IEEE Internet Computing*, 12(6):50–60, 2008.

[18] T. Li, J. Tang, and J. Xu. A predictive scheduling framework for fast and distributed stream data processing. In *Proc. of IEEE Big Data '15*, 2015.

[19] 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.

[20] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE Network*, 17(6), 2003.

[21] S. Rizou, F. Durr, and K. Rothermel. Solving the multi-operator placement problem in large-scale operator networks. In *Proc. of ICCCN '10*, 2010.

[22] I. Stanoi, G. Mihaila, T. Palpanas, and C. Lang. WhiteWater: distributed processing of fast streams. *IEEE Trans. Softw. Eng.*, 19(9):1214–1226, 2007.

[23] 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.

[24] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, et al. Storm@Twitter. In *Proc. of ACM SIGMOD '14*, pages 147–156, 2014.

[25] 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.

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

[27] M. Zaharia, M. Chowdhury, T. Das, A. Dave, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proc. of USENIX NSDI '12*, 2012.

[28] Y. Zhou, B. C. Ooi, K.-L. Tan, and J. Wu. Efficient dynamic operator placement in a locally distributed continuous query system. In *Proc. of OTM '06*, volume 4275 of *LNCS*, pages 54–71. Springer, 2006.

[29] Q. Zhu and G. Agrawal. Resource allocation for distributed streaming applications. In *Proc. of IEEE ICPP '08*, pages 414–421, 2008.