International Journal of Data Science and Analytics
https://doi.org/10.1007/s41060-018-0107-0

REVIEW

The many faces of data-centric workflow optimization: a survey

Georgia Kougka[1] · **Anastasios Gounaris**[1] · **Alkis Simitsis**[2]

Received: 26 May 2017 / Accepted: 24 February 2018

Abstract
Workflow technology is rapidly evolving and, rather than being limited to modeling the control flow in business processes, is becoming a key mechanism to perform advanced data management, such as big data analytics. This survey focuses on data-centric workflows (or workflows for data analytics or data flows), where a key aspect is data passing through and getting manipulated by a sequence of steps. The large volume and variety of data, the complexity of operations performed, and the long time such workflows take to compute give rise to the need for optimization. In general, data-centric workflow optimization is a technology in evolution. This survey focuses on techniques applicable to workflows comprising arbitrary types of data manipulation steps and semantic inter-dependencies between such steps. Further, it serves a twofold purpose: firstly, to present the main dimensions of the relevant optimization problems and the types of optimizations that occur before flow execution and secondly, to provide a concise overview of the existing approaches with a view to highlighting key observations and areas deserving more attention from the community.

Keywords Data flows · Workflow optimization · Workflow management systems · Data analysis · Data science

1 Introduction

Workflows aim to model and execute real-world intertwined or interconnected processes, named as *tasks* or *activities*. While this is still the case, workflows play an increasingly significant role in processing very large volumes of data, possibly under highly demanding requirements. Scientific workflow systems tailored to data-intensive e-science applications have been around since the last decade, e.g., [26,29]. This trend is nowadays complemented by the evolution of workflow technology to serve (big) data analysis, in settings such as business intelligence, e.g., [19], and business process management, e.g., [11]. Additionally, massively parallel engines, such as Spark, are becoming increasingly popular for designing and executing workflows.

Broadly, there are two big workflow categories, namely *control-centric* and *data-centric*. A workflow is commonly represented as a directed graph, where each task corresponds to a node in the graph and the edges represent the *control flow* or the *data flow*, respectively. The *control-centric workflows* are most often encountered in business process management [105], and they emphasize the passing of control across tasks and gateway semantics, such as branching execution, iterations, and so on; transmitting and sharing data across tasks is a second class citizen. In control-centric workflows, only a subset of the graph nodes correspond to activities, while the remainder denote events and gateways, as in the BPMN standard. In *data-centric workflows* (or workflows for data analytics or simply *data flows*[1]), the graph is typically acyclic (directed acyclic graph—DAG). The nodes of the DAG represent solely actions related to the manipulation, transformation, access and storage of data, e.g., as in [27,74,90,114] and in popular data flow systems, such as Pentaho Data Integration (Kettle) and Spark. The tokens passing through the tasks correspond to processed data. The control is modeled implicitly assuming that each task may start executing when the entire or part of the input becomes available. This survey considers data-centric flows exclusively.

✉ Anastasios Gounaris
gounaria@csd.auth.gr

Georgia Kougka
georkoug@csd.auth.gr

Alkis Simitsis
alkis@hpe.com

[1] Aristotle University of Thessaloniki, Thessaloníki, Greece

[2] HP Labs, Palo Alto, USA

[1] Hereafter, these three terms will be used interchangeably; the terms workflow and flow will be used interchangeably, too.

Published online: 06 March 2018

Executing data-centric flows efficiently is a far from trivial issue. Even in the most widely used data flow tools, flows are commonly designed manually. Problems in the optimality of those designs stem from the complexity of such flows and the fact that in some applications, flow designers might not be systems experts [2] and consequently, they tend to design with only semantic correctness in mind. In addition, executing flows in a dynamic environment may entail that an optimized design in the past may behave suboptimally in the future due to changing conditions [39,93].

The issues above call for a paradigm shift in the way data flow management systems are engineered and more specifically; there is a growing demand for automated optimization of flows. An analogy with database query processing, where declarative statements, e.g., in SQL, are automatically parsed, optimized, and then passed on to the execution engine is drawn. But data flow optimization is more complex, because tasks need not belong to a predefined set of algebraic operators with clear semantics and there may be arbitrary dependencies among their execution order. In addition, in data flows there may be optimization criteria apart from performance, such as reliability and freshness depending on business objectives and execution environments [89]. This survey covers optimization techniques[2] applicable to data flows, including database query optimization techniques that consider arbitrary plan operators, e.g., user-defined functions (UDFs), and dependencies between them. To the contrary, we do not aim to cover techniques that perform optimizations considering solely specific types of tasks, such as filters, joins and so on; the techniques covered in this survey do not necessarily rely on any type of algebraic task modeling.

The contribution of this survey is the provision of a taxonomy of data flow optimization techniques that refer to the flow plan generation layer. In addition, a concise overview of the existing approaches with a view to (i) explaining the technical details and the distinct features of each approach in a way that facilitates result synthesis; and (ii) highlighting strengths and weaknesses, and areas deserving more attention from the community is provided.

The main findings are that on the one hand, big advances have been made and most of the aspects of data flow optimization have started to be investigated. On the other hand, data flow optimization is rather a technology in evolution. Contrary to query optimization, research so far seems to be less systematic and mainly consists of ad hoc techniques, the combination of which is unclear.

The structure of the rest of this article is as follows. The next section describes the survey methodology and provides details about the exact context considered. Section 3 presents a taxonomy of existing optimizations that take place before the flow enactment. Section 4 describes the state-of-the-art techniques grouped by the main optimization mechanism they employ. Section 5 presents the ways in which optimization proposals for data-centric workflows have been evaluated. Section 6 highlights our findings. Section 7 touches upon tangential flow optimization-related techniques that have recently been developed along with scheduling optimizations taking place during flow execution. Section 8 reviews surveys that have been conducted in related areas, and finally, Sect. 9 concludes the paper.

2 Survey methodology

We first detail our context with regard to the architecture of a Workflow Management System (WfMS). Then, we explain the methodology for choosing the techniques included in the survey and their dimensions, on which we focus. Finally, we summarize the survey contributions.

2.1 Our context within WfMSs

The life cycle of a workflow can be regarded as an iteration of four phases, which cover every stage from the workflow modeling until its output analysis [71]. The four phases are *composition, deployment, execution*, and *analysis* [71]. The type of workflow optimization, on which this work focuses, is part of the deployment phase where the concrete executable workflow plan is constructed defining execution details, such as the engine that will execute each task. Additionally, Liu et al. [71] introduce a functional architecture for each data-centric Workflow Management System (WfMS), which consists of five layers: (i) *presentation*, which comprises the user interface; (ii) *user services*, such as the workflow monitoring and data provision components; (iii) *workflow execution plan (WEP) generation*, where the workflow plan is optimized, e.g., through workflow refactoring and parallelization, and the details needed by the execution engine are defined; (iv) *WEP execution*, which deals with the scheduling and execution of the (possibly optimized) workflow, but also considers fault-tolerance issues, and finally, (v) the *infrastructure* layer, which provides the interface between the workflow execution engine and the underlying physical resources.

According to the above architecture, one of the roles of a WfMS is to compile and optimize the workflow execution plans just before the workflow execution. Optimization of data flows, as conceived in this work, forms an essential part of the WEP generation layer and not of the execution layer. Although there might be optimizations in the WEP execution layer as well, e.g., while scheduling the WEP, these are out of our scope. More specifically, the mapping of flow tasks to concrete processing nodes during execution, e.g, task X

[2] The terms *technique, proposal*, and *work* will be used interchangeably.

of the flow should run on processing node Y, is traditionally considered to be a scheduling activity that is part of WEP execution layer rather than the WEP generation one, on which we focus. Finally, we use the terms task and activity interchangeably, both referring to entities that are not yet instantiated, activated or executed.

2.2 Techniques covered

The main part of this survey covers all the data flow optimization techniques that meet the following criteria to the best of authors' knowledge:

- They refer to the WEP generation layer in the architecture described above that is the focus is on the optimizations performed before execution rather than during execution.
- They refer to techniques that are applicable to any type of tasks rather than being tailored to specific types, such as filters and joins, or to an algebraic modeling of tasks.
- The partial ordering of the flow tasks is subject to dependency (or, else precedence) constraints between tasks, as is the generic case for example of scientific and data analysis flows; these constraints denote whether a specific task must precede another task or not in the flow plan.

We surveyed all types of venues where relevant techniques are published. Most of the covered works come from the broader data management and e-science community, but there are proposals from other areas, such as algorithms. We also include techniques that were proposed without generic data flows in mind, but meet our criteria and thus are applicable to generic data flows. An example is the proposal for queries over Web Services (WSs) in [94]. The main keywords we searched for are: "*workflow optimization*," "*flow optimization*," "*query optimization* AND *constraints*," and "*query optimization* AND *UDF*," while we applied snowballing in both directions [110] using both the reference list of and the citations to a paper.

2.3 Technique dimensions considered

We assume that the user initially defines the flow either at a high-level non-executable form or in an executable form that is not optimized. The role of the optimizations considered is to transform the initial flow into an optimized ready-to-be executed one.[3] Analogously to query optimization, it is convenient to distinguish between high-level and low-level flow details. The former capture essential flow parts, such as the final task sequencing, at a higher level than that of complete execution details, whereas the latter include all the information needed for execution. In order to drive the optimization, a set of metadata is assumed to be in place. This metadata can be statistics, e.g., cost per task invocation and size of task output per input data item, information about the dependency constraints between tasks, that is a partial order of tasks, which must be always preserved to ensure semantic correctness, or other types of information as explained in this survey.

To characterize optimizations that take place before the flow execution (or enactment), we pose a set of complementary questions when examining each existing proposal aiming at shedding light onto and covering all the main aspects of interest:

1. *What is the effect on the execution plan?*, which aims to identify the type of incurred enhancements to the initial flow plan.
2. *Why?*, which asks for the objectives of the optimization.
3. *How?*, which aims to clarify the type of the solution.
4. *When?*, to distinguish between cases where the WEP generation phase takes place strictly before the WEP execution one, and where these phases are interleaved.
5. *Where the flow is executed?*, which refers to the execution environment.
6. *What are the requirements?*, which refers to the input flow metadata in order to apply the optimization.
7. *In which application domain?*, which refers to the domain for which the technique initially targets.

We regard each of the above questions as a different dimension. As such, we derive seven dimensions: (i) the *Mechanisms* referring to the process through which an initial flow is transformed into an optimized one; (ii) the *Objectives* that capture the one or more criteria of the optimization process; (iii) the *Solution Types* defining whether an optimization solution is accurate or approximate with respect to the underlying formulation of the optimization problem; (iv) the *Adaptivity* during the flow execution; (v) the *Execution Environment* of the flow and its distribution; (vi) the *Metadata* necessary to apply the optimization technique; and finally, (vii) the *Application Domain*, for which each optimization technique is initially proposed.

[3] Through considering optimizations starting from a valid initial flow, we exclude from our survey the big area of answering queries in the presence of limited access patterns, in which, the main aim is to construct such an initial plan [69,78] through selecting an appropriate subset of tasks from a given task pool; however, we have considered works from data integration that optimize the plan after it has been devised, such as [111] or [34], which is subsumed by Kougka and Gounaris [60].

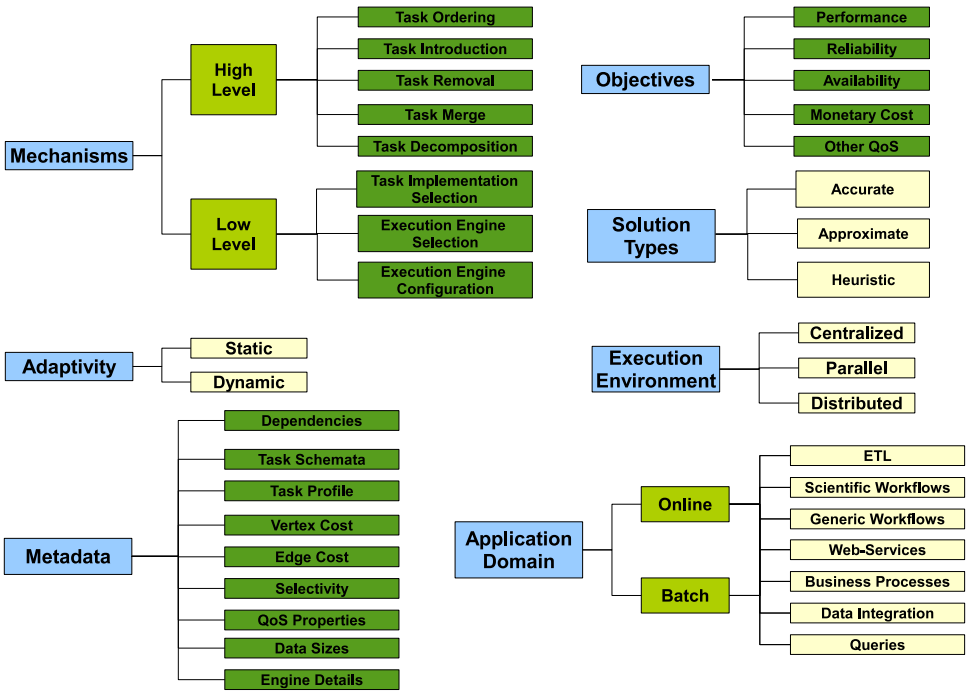

Fig. 1 A taxonomy of data-centric flow optimization for each of the identified dimensions

3 Taxonomy of existing solutions

Based on the dimensions identified above, we build a taxonomy of *existing* solutions. More specifically, for each dimension, we gather the values encountered in the techniques covered hereby. In other words, the taxonomy is driven by the current state-of-the-art and aims to provide a bird's eye view of today's data flow optimization techniques. The taxonomy is presented in Fig. 1 and analyzed below, followed by a discussion of the main techniques proposed to date in the next section. In the figure, each dimension (in light blue) can take one or more values. Single-value and multi-value dimensions are shown as yellow and green rectangles, respectively.

3.1 Flow optimization mechanisms

A data flow is typically represented as a directed acyclic graph (DAG) that is defined as $G = (V, E)$, where V denotes the nodes of the graph corresponding to a set of tasks and E represents a set of pair of nodes, where each pair denotes the data flow between the tasks. If a task outputs data that cannot be directly consumed by a subsequent task, then data transformation needs to take place through a third task; no data transformation takes place through an edge. Each graph element, either a vertex or an edge, is associated with properties, such as how exactly is implemented, for which execution engine, and under which configuration. Data flow optimization is a multi-dimensional problem, and its multiple dimensions are broadly divided according to the two flow specification levels. Consequently, we identify the optimization of the *high-level* (or *logical*) flow plan and the *low-level* (or *physical*) flow plan, and each type of optimization mechanism can affect the set of V or E of the workflow graph and their properties.

The logical flow optimization types are largely based on workflow structure reformations, while preserving any dependency constraints between tasks; structure reformations are reflected as modifications in V and E. The output of the optimized flow needs to be semantically equivalent as the output of the initial flow, which practically means that two flows receive the same input data and produce the same output data without considering the way this result was produced. Given that data manipulation takes place only in the context of tasks, logical flow optimization is task-oriented. The logical optimization types are characterized as follows (summarized also in Fig. 2):

- *Task Ordering*, where we change the sequence of the tasks by applying a set of partial (re)orderings.
- *Task Introduction*, where new tasks are introduced in the data flow plan in order, for example, to minimize the data to be processed and thus, the overall execution cost.
- *Task Removal*, which can be deemed as the opposite of task introduction. A task can be safely removed from the flow, if it does not actually contribute to its result dataset.
- *Task Merge* is the optimization action of grouping flow tasks into a single task without changing the semantics, for example, to minimize the overall flow execution cost or to mitigate the overhead of enacting multiple tasks.

– *Task Decomposition*, where a set of grouped tasks is split to more than one flow tasks with less complex functionality for generating more optimal sub-tasks. This is the opposite operation of merge action and may provide more optimization opportunities, as discussed in [47,90], because of the potential increase in the number of valid (re)orderings.

At the low level, a wide range of implementation aspects need to be specified so that the flow can be later executed (see also Fig. 3):

– *Task Implementation Selection*, which is one of the most significant lower-level problems in flow optimization. This optimization type includes the selection of the exact, logically equivalent, task implementation for each task that will satisfy the defined optimization objectives [90]. A well-known counterpart in database optimization is choosing the exact join algorithm (e.g., hash-join, sort-merge-join, nested loops).
– *Execution Engine Selection*, where we have to decide the type of processing engine to execute each task. The need for such optimization stems from the availability of multiple options in modern data-intensive flows [63,91]. Common choices, nowadays, include DBMSs, massively parallel engines, such as Hadoop clusters, apart from the execution engines that are bundled with data flow management systems.
– *Execution Engine Configuration*, where we decide on configuration details of the execution environment, such as the bandwidth, CPU, memory to be reserved during execution or the number of cores allocated [93].

The fact that the optimization types are task-oriented must not lead to a misinterpretation that they are unsuitable for data flows. Again, we draw an analogy with query optimization, where the main techniques, e.g., dynamic programming for join ordering, filter push down, and so on are operator-oriented; nevertheless, such an approach has proven sufficient for making query plans capable of processing terabytes of data.

3.2 Optimization objectives

An optimization problem can be defined as either *single* or *multiple objective* one depending on the number of criteria that considers. The optimization objectives that are typically presented in the state-of-the-art include the following: *performance*, *reliability*, *availability*, and *monetary cost*. The latter is important when the flow is executed on resources provided at a price, as in public clouds. Other quality metrics can be applied as well (denoted as *other QoS* in Fig. 1).

The first two objectives require further elaboration. Performance can be defined in several forms, depending, for example, on whether the target is the minimization of the response time, or the resource consumption. The formal definitions of the performance objective in data flows that have appeared in the literature are presented in the next section. Analogously, reliability may appear in several forms. In our context, reliability reflects how much confidence we have in a data flow execution plan to complete successfully. However, in data flow optimization proposals, we have also encountered the following two reliability aspects playing the role of optimization objectives: *trustworthiness* of a flow (Trust), which is typically based on the trustworthiness of the individual tasks and avoidance of dishonest providers, that is providers with bad reputation; and *Fault Tolerance*, which allows the execution of the flow to proceed even in the case of failures.

3.3 Optimization solution types

The optimization techniques that have been proposed constitute *accurate*, *approximate* or *heuristic* solutions. Such solutions make sense only when considered in parallel with the complexity of the exact problem they aim to solve. Unfortunately, a big set of the problems in flow optimization are intractable. For such problems, in the case of accurate solutions, a scalable technique cannot be provided. In the case of approximate optimization solutions, we typically tackle intractable problems in a scalable way while being able to provide guarantees on the approximation bound. Finally, in the last category, we exploit knowledge about the specific problem characteristics and propose algorithms that are fast and exhibit good behavior in test cases, without examining the deviation of the solution from the optimal in a formal manner.

3.4 Adaptivity of data-centric flow

Data flow adaptivity refers to the ability of technique to re-optimize the data flow plan during the execution phase. So, we characterize the optimization techniques as either *static*, where once the flow execution plan is derived, it is executed in its entirety, or *dynamic*, where the flow execution plan may be revised on the fly.

3.5 Execution environment

The techniques that are proposed for data flow optimization problem differ significantly according to the execution environment assumed. The execution environment is defined by the type of resources that execute the flow tasks. Specifically, in a *centralized execution environment*, all the tasks of a flow are executed by a single-node execution engine. Additionally,

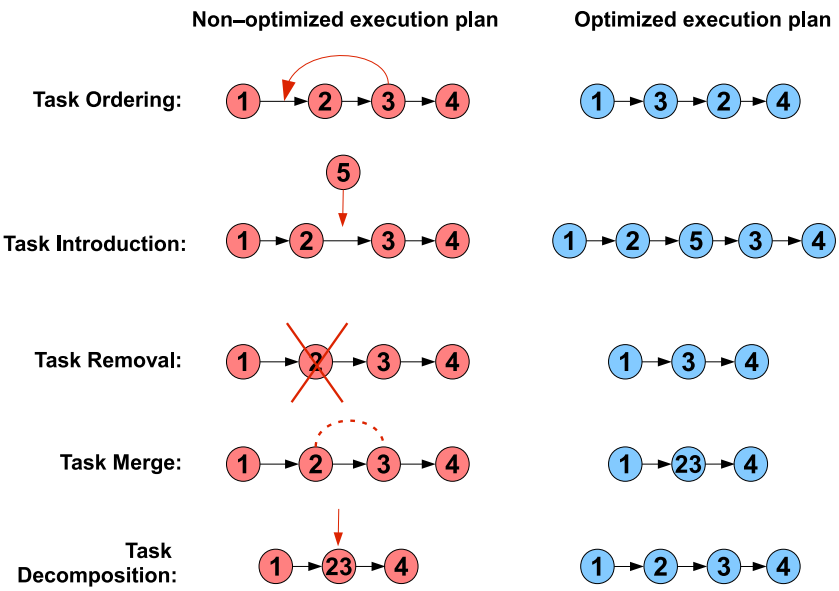

Fig. 2 Schematic representation of high-level flow optimizations

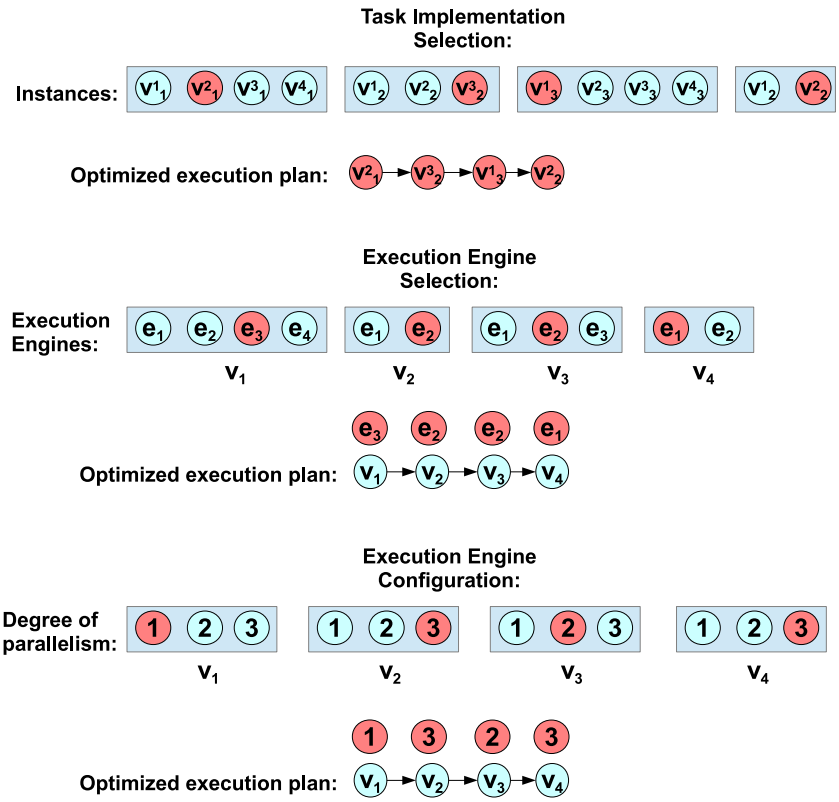

Fig. 3 Schematic representation of low-level flow optimizations

in a *parallel execution environment*, the tasks are executed in parallel by an engine on top of a homogeneous cluster, while in a *distributed execution environment*, the tasks are executed by remote and potentially heterogeneous execution engines, which are interconnected through ordinary network. Typically, optimizations on the logical level are agnostic to the execution environment, contrary to the physical optimization ones.

3.6 Metadata

The set of metadata includes the information needed to apply the optimizations and as such can be regarded as existential pre-conditions that should hold. The most basic input requirement of the optimization solutions is an initial set V of tasks. However, additional metadata with regard to the flow graph are typically required as well. These metadata are both

qualitative and quantitative (statistical), as discussed below. Qualitative metadata include:

- *Dependencies*, which explicitly refer to the definition of which vertices in the graph should always precede other vertices. Typically, the definition of dependencies comes in the form of an auxiliary graph.
- *Task schemata*, which refer to the definition of schema of the data input and/or output of each task. Note that dependencies may be produced by task schemata through simple processing [87], especially if they contain information about which schema elements are bound or free [58]. However, task schemata may serve additional purposes than deriving dependencies, e.g., to check whether a task contributes to the final desired output of the flow.
- *Task profile*, which refers to information about the execution logic of the task, that is the manner it manipulates its input data, e.g., through analysis of the commands implementing each task. If there are no such metadata, the task is considered as a black-box. Otherwise, information, e.g., about which attributes are read and which are written, can be extracted.

Quantitative metadata include:

- *Vertex cost*, which typically refers to the time cost, but can also capture other types of costs, such as monetary cost.
- *Edge cost*, which refers to the cost associated with edges, such as data transmission cost between tasks.
- *Selectivity*, which is defined as the (average) ratio of the output to the input data size of a task and its knowledge is equivalent to estimating the data sizes consumed and produced by each task; sizes are typically measured either in bytes or in number of records (cardinality).
- *QoS properties*, such as values denoting the task availability, reliability, security, and so on.
- *Engine details*, which cover issues, such as memory capacity, execution platform configurations, price of cloud machines, and so on.

3.7 Application domain

The final dimension across, which we classify existing solutions, is the application domain assumed when each technique is proposed. This dimension sheds light into differentiating aspects of the techniques with regard to the execution environment and the data types processed that cannot be captured by the previous dimensions. Note that the techniques may be applicable to arbitrary data flows in additional application domains than those initially targeted. In this dimension, we consider two aspects: (i) *domain* of initial proposal, which can be one of the following: ETL flows, data integration, Web Services (WSs) workflows, scientific workflows, MapReduce flows, business processes, database queries or generic; (ii) *online* (e.g., real time) versus *batch* processing. Generic domain proposals aim to a broader coverage of data flow applications, but due to their genericity, they make miss some optimization opportunities that a specific domain proposal could exploit. Also, online applications require more sophisticated solutions, since data are typically streaming and employ additional optimization objectives, such as reliability and acquiring responses under pressing deadlines.

4 Presentation of existing solutions

Here, we describe the main techniques grouped according to the optimization mechanism. This type of presentation facilitates result synthesis. Grouping by mechanism makes it easier to reason as to whether different techniques employing the same mechanism can be combined or not, e.g., because the make incompatible assumptions. Additionally, the solutions for each mechanism are largely orthogonal to the solutions for another mechanism, which means that, in principle, they can be combined at least in a naive manner. Therefore, our presentation approach provides more insights into how the different solutions can be synthesized.

The discussion is accompanied by a summary of each proposal in Table 1 for the dimensions of *mechanisms*, *objectives*, *solution types*, and *metadata*, and Table 2, for the *adaptivity*, *execution environment*, and *application domain* dimensions. When an optimization proposal comes in the form of an algorithm, we also provide the time complexity with respect to the size of the set of vertices $|V| = n$. However, the interpretation of such complexities requires special attention, when there are several other variables of the problem size, as is common in techniques employing optimization mechanisms at the physical level; details are provided within the main text. The first column of the table mentions also the publication year of each proposal, in order to facilitate the understanding of the proposal's setting and the time evolution of flow optimization.

Finally, we use a simple running example to present the application of the mechanisms. Specifically, as shown in Fig. 4, we consider a data flow that (i) retrieves Twitter posts containing product tags (*Tweets Input*), (ii) performs sentiment analysis (*Sentiment Analysis*), (iii) filters out tweets according to the results of this analysis ($Filter_1$), (iv) extracts the product to which the tweet refers to (*Lookup ProductID*), and (v) accesses a static external data source with additional product information (*Join* with *External Source*) in order to produce a report (*Report Output*). In this simple example, in any valid execution plan step (ii) should precede step (iii) and step (iv) should precede step (v).

Table 1 A summary of the main techniques for producing an optimized flow regarding the dimensions: *mechanisms*, *objectives*, *solution types*, and *metadata*

(References, years)	Mechanisms	Objectives	Solution types	Metadata
([1], 2008), ([48], 2007)	Merge, Engine Selection	Performance	Heuristic	Task Profile
([6], 2012)	Ordering	Performance (Bottleneck/ Critical Path)	Accurate ($O(n^6)$)	Dependencies, Vertex Cost, Selectivity
([15], 2008) extending [94]	Ordering, Implementation Selection	Performance	Heuristic	Dependencies, Task Schemata, Vertex Cost, Selectivity
([20], 1999)	Ordering	Performance (Sum Cost)	Approximate	Vertex Cost, Selectivity
([22], 2009)	Implementation Selection	Performance (Critical Path), Monetary Cost, Reliability	Heuristic	Vertex Cost, QoS properties
([24], 2014)	Removal	Performance	Heuristic ($O(n^2)$)	Task Schemata
([25], 2015)	Engine Configuration	Performance	Heuristic	Task profile
([30], 2005)	Removal	Performance	Heuristic	Dependencies, Task Schemata
([31], 2012)	Ordering	Performance (Throughput)	Accurate ($O(n^3)$)	Dependencies, Vertex Cost, Selectivity
([40], 1998)	Ordering	Performance (Sum Cost)	Approximate	Vertex Cost, Selectivity
([45], 2015)	Engine Configuration	Performance	Heuristic	Task Profile, Engine Details
([46], 2015) extending [44]	Task Introduction Engine Selection/ Configuration	Performance,Monetary Cost, Reliability (Fault Tolerance)	Accurate (exponential)	Vertex Cost, Engine Details
([47], 2012), ([81], 2015)	Ordering, Introduction/Removal, Decomposition	Performance (Sum Cost)	Accurate (exponential)	Task Schemata/Profile, Vertex Cost, Selectivity
([57], 2011)	Engine Configuration	Performance (Sum Cost), Monetary Cost	Heuristic	Vertex Cost
([60], 2017), ([59], 2014)	Ordering	Performance (Sum Cost)	Accurate (exponential), Approximate ($O(n^2)$)	Dependencies, Vertex Cost, Selectivity
([63], 2014)	Engine Selection	Performance (Sum Cost)	Heuristic ($O(n)$)	Dependencies Vertex/Edge Cost

Table 1 continued

(References, years)	Mechanisms	Objectives	Solution types	Metadata
([65], 2010)	Ordering	Performance (Sum Cost)	Approximate ($O(n^2)$)	Task Schemata, Vertex Cost, Selectivity
([67], 2013)	Implementation Selection, Engine Configuration	Performance, Other QoS	Heuristic ($O(n)$)	Vertex Cost, QoS properties
([68], 2008)	Implementation Selection	Performance, Availability, Monetary Cost	Heuristic ($O(n)$)	Vertex Cost, y QoS properties
([70], 2012)	Merge, Engine Configuration	Performance	Heuristic	Vertex Cost, Task Schemata, Selectivity, Engine Details
([72], 2015)	Engine Configuration	Performance	Heuristic	Vertex Cost, Task Profile
([84], 2014)	Engine Configuration	Performance	Exhaustive	Vertex Cost, Engine Details
([87], 2005)	Ordering, Merge	Performance (Sum Cost)	Accurate (exponential), Heuristic ($O(n^2)$)	Vertex Cost, Task Schemata
([90], 2012), ([91], 2013), ([93], 2013)	Ordering, Decomposition, Engine/Implementation Selection	Performance (Constr. Sum Cost Bottleneck), Reliability (Fault Tolerance)	Accurate (exponential), Heuristic ($O(n^2)$)	Task Schemata, Vertex Cost
([92], 2010) extending [87]	Ordering, Merge, Introduction, Implementation Selection, Engine Configuration	Performance (Constr. Sum Cost Bottleneck), Reliability (Fault Tolerance)	Heuristic ($O(n^2)$)	Task Schemata, Vertex Cost
([94], 2006)	Ordering	Performance (Bottleneck)	Accurate ($O(n^5)$)	Dependencies, Vertex Cost, Selectivity
([95], 2012)	Implementation Selection	Performance, Monetary Cost, Reliability	Heuristic ($O(n)$)	Vertex Cost
([98,99], 2011)	Ordering	Performance (Bottleneck)	Heuristic (exponential)	Dependencies, Vertex/Edge Cost, Selectivity
([101], 2007)	Implementation Selection, Task Introduction	Performance (Sum Cost)	Accurate (exponential)	Vertex cost
([107], 2007)	Merge	Performance	Heuristic	Task Profile
([108], 2005)	Implementation Selection	Performance, Availability, Reliability (Trust)	Heuristic ($O(n)$)	Vertex Cost, QoS properties
([111], 1999)	Ordering	Performance (Sum Cost)	Approximate ($O(n^2)$)	Task Schemata, Vertex Cost
([113], 2015)	Engine Selection	Performance, Monetary Cost	Heuristic	Vertex Cost, Engine details

Table 2 A summary of the main techniques for producing an optimized flow regarding the dimensions: *adaptivity*, *execution environment*, and *application domain*

(References, years)	Adaptivity		Execution environment			Application domain
	Static	Dynamic	Centralized	Parallel	Distributed	
([1], 2008), ([48], 2007)	★	–	★	–	–	ETL (Batch)
([6], 2012)	★	–	–	★	–	Queries (Online)
([15], 2008) extending [94]	★	–	–	–	★	Web Services (Online)
([20], 1999)	★	–	★	–	–	Queries (Batch)
([22], 2009)	★	–	–	–	★	Web Services (Batch)
([24], 2014)	★	–	★	–	–	Scientific Workflows (Batch)
([25], 2015)	★	–	–	★	–	Generic
([30], 2005)	–	★	–	★	–	Scientific Workflows (Batch)
([31], 2012)	★	–	–	★	–	Queries (Online)
([40], 1998)	★	–	★	–	–	Queries (Batch)
([45], 2015)	–	★	–	★	–	Map Reduce (Batch)
([46], 2015) extending [44]	★	–	–	–	★	Scientific Workflows (Batch)
([47], 2012), ([81], 2015)	★	–	–	★	–	Scientific Workflows (Batch)
([57], 2011)	★	–	–	–	★	Scientific (Online)
([60], 2017), ([59], 2014)	★	–	★	–	–	Generic
([63], 2014)	★	–	–	–	★	Generic
([65], 2010)	★	–	★	–	–	ETL (Batch)
([67], 2013)	–	★	–	–	★	Generic
([68], 2008)	★	–	–	–	★	Web Services (Online)
([70], 2012)	★	–	–	★	–	Map Reduce (Batch)
([72], 2015)	★	–	–	★	–	ETL (Batch)
([84], 2014)	★	–	–	★	–	MapReduce (Batch)
([87], 2005)	★	–	★	–	–	ETL (Batch)
([90], 2012), ([91], 2013), ([93], 2013)	★	–	–	–	★	ETL (Online)
([92], 2010) extending [87]	★	–	–	★	–	ETL (Online)
([94], 2006)	★	–	–	–	★	Web Services (Online)
([95], 2012)	★	–	–	–	★	Generic
([98,99], 2011)	★	–	–	–	★	Web Services (Online)
([101], 2007)	★	–	★	–	–	ETL (Batch)
([107], 2007)	★	–	–	★	–	Business Processes (Batch)
([108], 2005)	★	–	–	–	★	Web Services (Online)
([111], 1999)	★	–	–	–	★	Data Integration (Online)
([113], 2015)	★	–	–	–	★	Generic

4.1 Task ordering

The goal of *Task Ordering* is typically specified as that of optimizing an objective function, possibly under certain constraints. A common feature of all proposals is that they assign a metric $m(v_i)$ to each vertex $v_i \in V, i = 1 \ldots n$. To date, task ordering techniques have been employed to optimize performance. More specifically, all aspects of performance that we introduced previously have been investigated: the minimization of the sum of execution costs of either all tasks (both under and without constraints) or the tasks that belong to the critical path, the minimization of the maximum task cost, and the maximization of the throughput. Table 3 summarizes the objective functions of these metrics that have been employed by approaches to task ordering in data flow optimization to date. Existing techniques can be modeled at an abstract level uniformly as follows. The metric m refers either to costs (denoted as $c(v_i)$) or to throughput values (denoted as $f(v_i)$). Costs are expressed in either time or abstract units, whereas throughput is expressed as number of records (or tuples) processed per time unit. A more generic

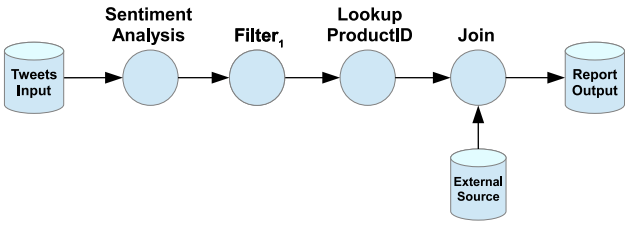

Fig. 4 A data flow processing Twitter posts

modeling assigns a cost to each vertex v_i along with its outcoming edges e_{ij}, $j = 1 \ldots n$ (denoted as $c(v_i, e_{ij})$).

These objective functions correspond to problems with different algorithmic complexities. Specifically, the problems that target the minimization of the sum of the vertex cost are intractable [16]. Moreover, Burge et al. [16] discuss that *"it is unlikely that any polynomial time algorithm can approximate the optimal plan to within a factor of* $O(n^{\theta})$*,"* where θ is some positive constant. The generic bottleneck minimization problem is intractable as well [97]. However, the bottleneck minimization based only on vertex costs and the other two objective functions can be optimally solved in polynomial time [5,31,94].

Independently of the exact optimization objectives, all the known optimization techniques in this category assume the existence of dependency constraints between the tasks either explicitly or implicity through the definition of task schemata. For the cost or throughput metadata, some techniques rely on the existence of lower-level information, such as selectivity (see Sect. 4.1.5).

4.1.1 Techniques for minimizing the sum of costs

Regarding the minimization of the sum of the vertex costs (first row in Table 3), there have been proposed both accurate and heuristic optimization solutions dealing with this intractable problem; apparently the former are not scalable. An accurate task ordering optimization solution is the application of the dynamic programming; dynamic programming is extensively used in query optimization [83] and such a technique has been proposed for generic data flows in [59]. The rationale of this algorithm is to calculate the cost of task subsets of size n based on subsets of size $n-1$. For each of these subsets, we keep only the optimal solution that satisfies the dependency constraints. This solution has exponential complexity even for simple linear non-distributed flows ($O(2^n)$) but, for small values of n, is applicable and fast.

Another optimization technique is the exhaustive production of all the topological sortings in a way that each sorting is produced from the previous one with the minimal amount of changes [102]; this approach has been also employed to optimize flows in [59,60]. Despite having a worst-case complexity of $O(n!)$, it is more scalable than dynamic programming solution, especially, for flows with many dependency constraints between tasks.

Another exhaustive technique is to define the problem as a state space search one [87]. In such a space, each possible task ordering is modeled as a distinct state and all states are eventually visited. Similar to the optimization proposals described previously, this technique is not scalable either. Another form of task re-ordering is when a single input/output task is moved before or after a multi-input or a multi-output task [87,92]. An example case is when two copies of a proliferate single input/ output task are originally placed in the two inputs of a binary fork operation and after re-ordering, are moved after the fork. In such a case, the two task copies moved downstream are merged into a single one. As another example, a single input/output task placed after a multi-input task can be moved upstream, e.g., when a filter task placed after a binary fork is moved upstream to both fork input branches (or to just one, based on their predicates). This is similar to traditional query optimization where a selective operation can be moved before an expensive operation like a join.

The branch-and-bound task ordering technique is similar to the dynamic programming one in that it builds a complete flow by appending tasks to smaller sub-flows. To this end, it examines only sub-flows in terms of meeting the dependency constraints and applies a set of recursive calls until generating all the promising data flow plans employing early pruning. Such an optimization technique has been applied in [47,81] for executing parallel scientific workflows efficiently, as part of a new optimization technique for the development of a logical optimizer, which is integrated into the Stratosphere system [8], the predecessor of Apache Flink. An interesting feature of this approach is that following common practice from database systems it performs static task analysis (i.e., task profiling) in order to yield statistics and fine-grained dependency constraints between tasks going further from the knowledge that can be derived from simply examining the task schemata.

For practical reasons, the four accurate techniques described above are not a good fit for medium and large flows, e.g., with over 15–20 tasks. In these cases, the space of possible solutions is large and needs to be pruned. Thus, heuristic algorithms have been presented to find near optimal solutions for larger data flows. For example, Simitsis et al. [87] propose a technique of task ordering by allowing state transitions, which corresponds to orderings that differ in the ordering of only two adjacent tasks. Such transitions are equivalent to a heuristic, which swaps every pair of adjacent tasks, if this change yields lower cost, always preserving the defined dependency constraints, until no further changes can be applied. This heuristic, initially proposed for ETL flows, can be applied to parallel and distributed execution environments with streaming or batch input data. Interestingly, this technique is combined with another set of heuristics using

additional optimization techniques, such as *task merge*. In general, this heuristic is shown to be capable of yielding significant improvements. Its complexity is $O(n^2)$, but there can be no guarantee for how much its solutions can deviate from the optimal one.

There is another family of techniques minimizing the sum of the tasks by ordering the tasks based on their rank value defined as $\frac{1-sel(v_i)}{c(v_i)}$, where $sel(v_i)$ is the selectivity of v_i. The first examples of these techniques were initially proposed for optimizing queries containing UDFs, while dependency constraints between pairs of a join and UDF are considered [20,40]. However, they can be applied in data flows by considering flow tasks as UDFs and performing straightforward extensions. For example, an extended version of [20], also discussed in [59], builds a flow incrementally in n steps instead of starting from a complete flow and performing changes. In each step, the next task to be appended is the one with the maximum rank value, for which all the prerequisite tasks have been already included. This results in a greedy heuristic of $O(n^2)$ time complexity.

This heuristic has been extended by Kougka et al. [60] with techniques that leverage the query optimization algorithm for join ordering by Krishnamurthy et al. [64] with appropriate post-processing steps in order to yield novel and more efficient task ordering algorithms for data flows. In [65], a similar rationale is followed with the difference that the execution plan is built from the sink to source task. Both proposals build linear plans, i.e., plans in the form of a chain with a single source and a single sink. These proposals for generic or traditional ETL data flows are essentially similar to the *Chain* algorithm proposed by Yerneni et al. [111] for choosing the order of accessing remote data sources in online data integration scenarios. Interestingly, in [111], it is explained that such techniques are n-competitive, i.e., they can deviate from the optimal plan up to n times.

The incurred performance improvements can be significant. Consider the example in Fig. 4, where let the cost per single input tweet of the five steps be 1, 10, 1, 1, and 5 units, respectively. Let the selectivities be 1, 1, 0.1, 1, and 0.15, respectively. Then, the average cost in Fig. 4 for each initial tweet is $1 + 10 + 1 + 0.1 + 0.5 = 12.6$, whereas the cost of the flow in Fig. 5 is $1+1+5+1.5+0.15 = 7.65$. In general,

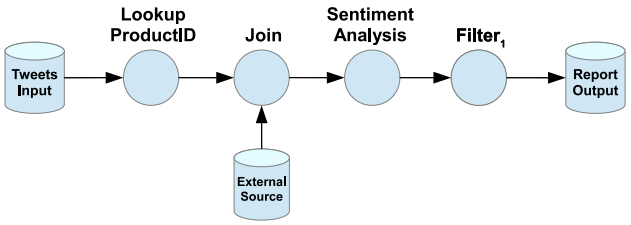

Fig. 5 An example of optimized task ordering

for ordering arbitrary flow tasks in order to minimize the sum of the task costs, any of the above solutions can be used. If the flow is small, exhaustive solutions are applicable [61]; otherwise, the techniques in [60] are the ones that seem to be capable of yielding the best plans.

Finally, minimizing the sum of the tasks cost appears also in multi-criteria proposals that consider also reliability, and more specifically fault tolerance [90,92]. These proposals employ a further constraint in the objective function denoted as function $g()$ (see second row in Table 3). In these proposals, $g()$ defines the number of faults that can be tolerated in a specific time period. The strategy for exploring the search space of different orderings extends the techniques that proposed by Simitsis et al. [87].

4.1.2 Techniques for minimizing the bottleneck cost

Regarding the problem of minimizing the maximum task cost (third row in Table 3), which acts as the performance bottleneck, there is a *Task Ordering* mechanism initially proposed for the parallel execution of online WSs represented as queries [94]. The rationale of this technique is to push the selective flow tasks (i.e., those with $sel < 1$) in an earlier stage of the execution plan in order to prune the input dataset of each service. Based on the selectivity values, there may be cases where the output of a service may be dispatched to multiple other services for executing in parallel or in a sequence having time complexity in $O(n^5)$ in the worst case. The problem is formulated in a way that it is tractable and the solutions is accurate.

Another optimization technique that considers task ordering mechanism for online queries over Web Services appears in [98,99]. The formulation in these proposals extends the

Table 3 A summary of the objective functions in task ordering

Description	Objective functions	References
Sum cost	min $\sum c(v_i)$, where $i = 1 \ldots n$	[47,59,65,81,87,111]
Constrained sum cost	min $\sum c(v_i)$, where $i = 1 \ldots n$ and $g(v_i) < 0$	[90–93]
Bottleneck cost	min $max(c(v_i))$, where $i = 1 \ldots n$	[5,6,94]
	min $max(c(v_i, e_{ij}))$, where $i = 1 \ldots n$	[98,99]
Critical path cost	min $\sum c(v_i)$, where v_i belongs to *critical path*	[5,6]
Throughput	max $\sum f(v_i)$, where $i = 1 \ldots n$	[31]

one proposed by Srivastava et al. [94] in that it considers also edge costs. This modification renders the problem intractable [97]. The practical value is that edge costs naturally capture the data transmission between tasks in a distributed setting. The solution proposed by Tsamoura et al. [98,99] consists of a branch-and-bound optimization approach with advanced heuristics for early pruning and despite its exponential complexity, it is shown that it can apply to flows with hundreds of tasks, for reasonable probability distributions of vertex and edge costs.

The techniques for minimizing the bottleneck cost can be combined with those for the minimization of the sum of the costs. More specifically, the pipelined tasks can be grouped together and for the corresponding sub-flow, the optimization can be performed according to the bottleneck cost metric. Then, these groups of tasks can be optimized considering the sum of their costs. This essentially leads to a hybrid objective function that aims to minimize the sum of the costs for segments of pipelining operators, where each segment cost is defined according to the bottleneck cost. A heuristic combining the two metrics has appeared in [92].

4.1.3 Techniques for optimizing the critical path

A technique that considers the critical path providing an accurate solution has appeared in [6]. This work has $O(n^6)$ time complexity and has been initially proposed for online queries in parallel execution environments, but is also applicable to data flows. The strong point of this solution is that it can perform bi-objective optimization combining the bottleneck and the critical path criteria.

4.1.4 Techniques for maximizing the throughput

Re-ordering the filter operators of a workflow can be used to find an optimal query execution plan that maximizes throughput leveraging pipelined parallelism. Such a technique has been presented by Deshpande et al. [31] considering queries with tree-shaped constraints for parallel execution environment providing an accurate solution that has $O(n^3)$ time complexity. In this proposal, each task is assumed to be executed on a distinct node, where each node has a certain throughput capacity that should not be exceeded. The unique feature of this proposal is that it produces a set of plans that need to be executed concurrently in order to attain throughput maximization. The drawback is that it cannot handle arbitrary constraint graphs, which implies that its applicability to generic data flows is limited.

4.1.5 Task cost models

Orthogonally to the objective functions in Table 3, different cost models can be employed to derive $c(v_i)$, the cost of the ith task v_i. The important issue is that a task cost model can be used as a component in any cost-based optimization technique, regardless of whether it has been employed in the original work proposing that technique. A common assumption is that $c(v_i)$ depends on the volume of data processed by v_i, but this feature can be expressed in several ways:

- $c(v_i) = \prod_{j=1}^{|T_i^{prec}|} sel_j * cpi_i$: this cost model defines the cost of the ith task as the product of (i) the cost per input data unit (cpi_i) and (ii) the product of the selectivities sel of preceding tasks; T_i^{prec} is the set of all the tasks between the data sources and v_i. This cost model is explicitly used in proposals such as [58–60,65,111].
- $c(v_i) = rs(v_i)$: In this case, the cost model is defined as the size of the results (rs) of v_i; it is used in [111], where each task is a remote database query.
- $c(v_i) = \alpha_i \cdot CPU(v_i) + \beta_i \cdot IO(v_i) + \gamma_i \cdot Ship(v_i)$: this cost model is a weighted sum of the three main cost components, namely the cpu, I/O, and data shipping costs. Further, $CPU(v_i)$ can be elaborated and specified as $\prod_{j=1}^{|T_i^{prec}|} sel_j * cpi_i$ (defined above) plus a startup cost. I/O costs depends on the cost per input data unit to access secondary storage. Data communication cost $Ship(v_i)$ depends on the size of the input of v_i, which, as explained earlier, depends also on previous tasks and the vertex selectivity sel_i. α, β, and γ are the weights. Such an elaborate cost model has been employed by Hueske et al. [47].
- $c(v_i) = proc(v_i) + part(v_i)$: This cost model is suggested by Simitsis et al. [92]. It explicitly covers task parallelization and splits the cost of a tasks into the processing cost $proc$ and the cost to partition and merge data $part$. The former cost is divided into a part that depends on input size and a fixed one. The proposal in [92] considers the tasks in the flow that add recovery points or create replicas by providing differently specific formulas for them.

4.1.6 Additional remarks

Regarding the execution environment, since the task (re-) ordering techniques refer to the logical WEP level, they can be applied to both centralized and distributed flow execution environments. However, in parallel and distributed environments, the data communication cost needs to be considered. The difference between these environments with regard to the communication cost is that in the latter, this cost depends both on the sender and receiver task and as such, it needs to be represented, not as a component of vertex cost but as a property of edge cost.

Additionally, very few techniques, e.g., [87], explicitly consider re-orderings between single input/output and

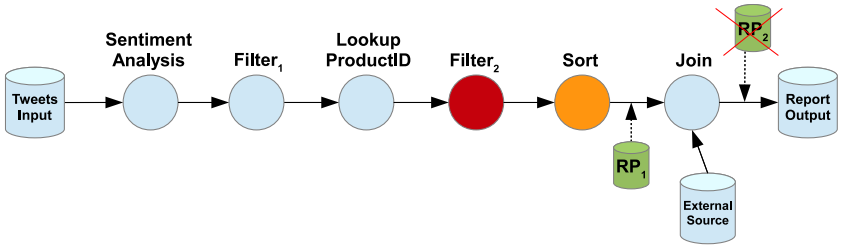

Fig. 6 Examples of *Task Introduction* techniques

multiple-input or multiple-output tasks; however, this type of optimization requires further investigation in the context of complex flow optimization.

Finally, none of the proposed techniques for task ordering technique discussed are adaptive ones, that is they do not consider workflow re-optimization during its execution phase. In general, adaptive flow optimization is a subarea in its infancy. However, Böhm et al. [13] have proposed solutions for choosing when to trigger re-optimization, which, in principle, can be coupled with any cost-based flow optimization technique.

4.2 Task introduction

Task introduction has been proposed for three reasons.

Firstly, to achieve fault-tolerance through the introduction of recovery points and replicator tasks in online ETLs [92]. For recovery points, a new node storing the current flow state is inserted in the flow in order to assist recovering from failures without needing to recompute the flow from scratch. Adding a recovery (to a specific point in the plan) depends on a cost function that compares the projected recovery cost in case of failure against the cost to maintain a recovery point. Additionally, the replicator nodes produce copies of specified sub-flows in order to tolerate local failures, when no recovery points can be inserted, e.g., because the associated overhead increases the execution time above a threshold. In both cases of task introduction, the semantics of the flow are immutable. The proposed technique extends the state space search in [87] after having pruned the state search space. The objective function employed is the constrained sum cost one (2nd row in Table 3), where the constraint is on the number of places where a failure can occur. The cost model explicitly covers the recovery maintenance overhead (last case in Sect. 4.1.5). The key idea behind the pruning of search space is first to apply task re-ordering and then, to detect all the promising places to add the recovery points based on heuristic rules. An example of the technique is in Fig. 6 and suppose that we examine the introduction of up to two recovery points. The two possible places are just after the *Sort* and *Join* tasks, respectively. Assume that the most beneficial place is the first one, denoted as RP_1. Also, given RP_1, RP_2 is discarded because it incurs higher cost than re-executing the *Join* task. Similarly to the recovery points above, the technique proposed by Huang et al. [46] introduces operations that copy intermediate data from transient nodes to primary ones, using a cluster of machines containing both transient and primary cloud machines; the former can be reclaimed by the cloud provided at any time, whereas the latter are allocated to flow execution throughout its execution.

Secondly, task introduction has been employed by Rheinländer et al. [81] to automatically insert explicit filtering tasks, when the user has not initially introduced them. This becomes plausible with a sophisticated task profiling mechanism employed in that proposal, which allows the system to detect that some data are not actually needed. The goal is to optimize a sum cost objective function, but the technique is orthogonal to any objective function aiming at performance improvement. For example, in Fig. 6, we introduce a filtering task if the final report needs only a subset of the initial data, e.g., it refers to a specific range of products.

Third, task introduction can be combined with *Implementation Selection* (Sect. 4.6). An example appears in [101], where the purpose is to exploit the benefit of processing sorted records. To this end, it explores the possibility of introducing new vertices, called sorters, and then to choose task implementations that assume sorted input; the overhead of the insertion of the new tasks is outweighed by the benefits of sort-based implementations. In Fig. 6, we add such a sorter task just before the *Join* if a sort-based join implementation and report output is preferred. Proactively ordering data to reduce the overall cost has been used in traditional database query optimization [35] and it seems to be profitable for ETL flows as well.

Finally, all these three techniques can be combined; for example, in the example all can apply simultaneously yielding the complete plan in the figure.

4.3 Task removal

A set of optimization proposals support the idea of removing a task or a set of tasks from the workflow execution plan without changing the semantics in order to improve the performance; these proposals have been proposed mostly for offline scientific workflows, where it is common to reuse tasks or sub-flows from previous workflows without necessarily examining whether all tasks included are actually necessary or whether some results are already present. Three

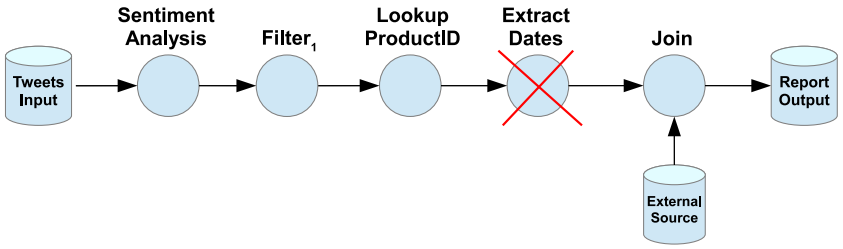

Fig. 7 An example of the *Task Removal* technique

techniques adopt this rationale [24,30,81], which are discussed in turn.

The idea of Rheinländer et al. [81] is to remove a task or multiple tasks until the workflow consists only of tasks that are necessary for the production of the desired output. This implies that the execution result dataset remains the same regardless of the changes that have been applied. It aims to protect users that have carelessly copied data flow tasks from previous flows. In Fig. 7, we see that, initially, the example data flow contains an *Extract Dates* task, which is not actually necessary.

The heuristic of Deelman et al. [30] has been proposed for a parallel execution environment and is one of the few dynamic techniques allowing the re-optimization of the workflow during the workflow execution. At runtime, it checks whether any intermediate results already exist at some node, thus making part of the flow obsolete. Both [81] and [30] are rule-based and do not target an objective function directly.

Another approach for applying task removal optimization mechanism is to detect the duplicate tasks, i.e., tasks performing exactly the same operation and keep only a single copy in the execution workflow plan [24]. This might be caused by carelessly combining existing smaller flows from a repository, e.g., myExperiment.[4] A necessary condition in order to ensure that there will be no precedence violations is that these tasks must be dependency constraint free, which is checked with the help of the task schemata. Such a heuristic has $O(n^2)$ time complexity.

4.4 Task merge

Task Merge has been also employed for improving the performance of the workflow execution plan. The main technique is to apply re-writing rules to merge tasks with similar functions into one bigger task. There are three techniques in this group, all tailored to a specific setting. As such, it is unclear whether they can be combined.

First, in [107], tasks that encapsulate invocations to an underlying database are merged so that fewer (and more complex) invocations take place. This rule-based heuristic has been proposed for business processes, for which it is common to access various data stores, and such invocations incur a large time overhead.

Second, a related technique has been proposed for SQL statements in commercial data integration products [1,48]. The rationale of this idea is to group the SQL statements into a bigger query in order to push the task functionalities to the best processing engine. Both approaches presented in [1,48] derive the necessary information about the functionality of each task with the help of task profiling and produce larger queries employing standard database technology. For example, instead of processing a series of SQL queries to transform data, it is preferable to create a single bigger query. As previously, the proposed optimization is a heuristic that does not target to optimize any objective function explicitly. A generalization of this idea to languages beyond SQL is presented by Simitsis et al. [90,93], and a programming language translator has been described by Jovanovic et al. [53,54].

Third, Harold et al. [70] presents a heuristic non-exhaustive solution for merging MapReduce jobs. Merging occurs at two levels: first MapReduce jobs are tried to be transformed into Map-only jobs. Then, sharing common Map or Reduce tasks is investigated. These two aspects are examined with the help of a 2-phase heuristic technique.

Finally, in the optimizations in [87,92], which rely on a state space search as described previously, adjacent tasks that should not be separated may be grouped together during optimization. The aim of this type of merger is not to produce a flow execution plan with fewer and more complex tasks (i.e., no actual task merge optimization takes place), but to reduce the search space so that the optimization is speeded up; after optimization, the merged tasks are split.

4.5 Task decomposition

An advanced optimization functionality is *Task Decomposition*, according to which, the operations of a task are split into more tasks, this results in a modification of the set V of vertices. This mechanism has appeared in [47,81] as a pre-processing step, before the task ordering takes place. Its advantage is that it opens up opportunities for ordering, i.e., it does not optimize an objective function in its own, but it enables more profitable task orderings.

Task decomposition is also employed by Simitsis et al. [90,91,93]. In these proposals, complex analysis tasks, such

[4] www.myexperiment.org/ in bio-informatics.

as sentiment analysis presented in previous examples, can be split into a sequence of tasks at a finer granularity, such as tokenization, and part-of-speech tagging.

Note that both these techniques are tightly coupled to the task implementation platform assumed.

4.6 Task implementation selection

A set of optimization techniques target the *Implementation Selection* mechanism. At a high level, the problem is that there exist multiple equivalent candidate implementations for each task and we need to decide which ones to employ in the execution plan. The issue of whether the different implementations may produce different results is orthogonal to this discussion as far as all implementations are acceptable by the user; however, we mostly refer to settings where equivalence implies also the production of the same result set. For example, a task encapsulating a call to a remote WS can contact multiple equivalent WSs, or a task may be implemented to run either in a single-threaded or in a multi-threaded mode. These techniques typically require as input metadata the vertex costs of each task implementation alternative. Suppose that, for each task, there are m alternatives. This leads to a total of $O(m^n)$ of combinations; thus, a key challenge is to cope with the exponential search space. In general, the number of alternatives for each task may be different and the total number of combinations is the product of these numbers. For example, in Fig. 8, there are four and three alternatives $(Impl_1, \ldots, Impl_n)$ for the *Sentiment Analysis* and *Lookup Product* tasks, respectively, corresponding to twelve combinations.

It is important to note that, conceptually, the choice of the implementation of each task is orthogonal to decisions on task ordering and the rest of the high-level optimization mechanisms. As such, the techniques in this section can be combined with techniques from the previous sections.

A brute force, and thus of exponential complexity approach to finding the optimal physical implementation of each flow task before its execution has appeared in [101]. This approach models the problem as a state space search one and, although it assumes that the sum cost objective function is to be optimized, it can support other objective functions too. An interesting feature of this solution is that it explicitly explores the potential benefit from processing sorted data. Also, the ordering and task introduction algorithm in [92] allows for choosing parallel flavors of tasks. The parallel flavors, apart from cloning the tasks as many times as the degree of partitioned parallelism decided, explicitly consider issues, such as splitting the input data, distributing them across all clones, and merging all their outputs. These issues are reflected in an elaborate cost function as mentioned previously, which is used to decide whether parallelization is beneficial.

Additionally to the optimization techniques above, there is a set of multi-objective optimization approaches for *Implementation Selection*. These multi-objective heuristics, apart from the vertex cost, require further metadata that depend on the specified optimization objectives. For example, several multi-objective optimization approaches have been proposed for flows, where each task is essentially an invocation to an online WS that may not be always available; in such settings, the aim of the optimizer is the selection of the best service for each service type taking into account both performance and availability metadata.

Three proposals that target this specific environment are [68,95,108]. To achieve scalability, each task is checked in isolation, thus resulting in $O(nm)$ time complexity, but at the expense of finding local optimal solutions only. Kyriazis et al. [68] consider availability, performance, and cost for each task. As initial metadata, scalar values for each objective and for candidate services are assumed to be in place. The main focus of the proposed solution is (i) on normalizing and scaling the initial values for each of the objectives and (ii) on devising an iterative improvement algorithm for making the final decisions for each task. The multi-objective function is either the optimization of a single criterion under constraints on the others or the optimization of all the objectives at the same time. However, in both cases, no optimality guarantees (e.g., finding a Pareto optimal solution) are provided.

The proposal in [108] is similar in not guaranteeing pareto optimal solutions. It considers performance, availability, and reliability for each candidate WS, where each criterion is weighted and contributes to a single scalar value, according to which services are ordered. The notion of reliability in this proposal is based on its trustworthiness. [95] is another service selection proposal that considers the three objectives, namely performance, monetary cost, and reliability in terms of successful execution. The service metadata are normalized, and the technique proposed employs a max-min heuristic that aims to select a service based on its smallest normalized value. An additional common feature of the proposals in [68,95,108] is that no objective function is explicitly targeted.

Another multi-objective optimization approach to choosing the best implementation selection of each task consists of linear complexity heuristics [67]. The main value of those heuristics are that they are designed to be applied on the fly, thus forming one of the few existing adaptive data flow optimization proposals. Additionally, the technique proposed by Braga et al. [15] extends the task ordering approach in [94] so that, for each task, the most appropriate implementation is first selected. None of these proposals employ a specific objective function as well. Finally, multi-objective WS selection mechanism can be performed with the help of ant colony optimization algorithms; an example of applying this optimization technique for selecting WS instantiations between

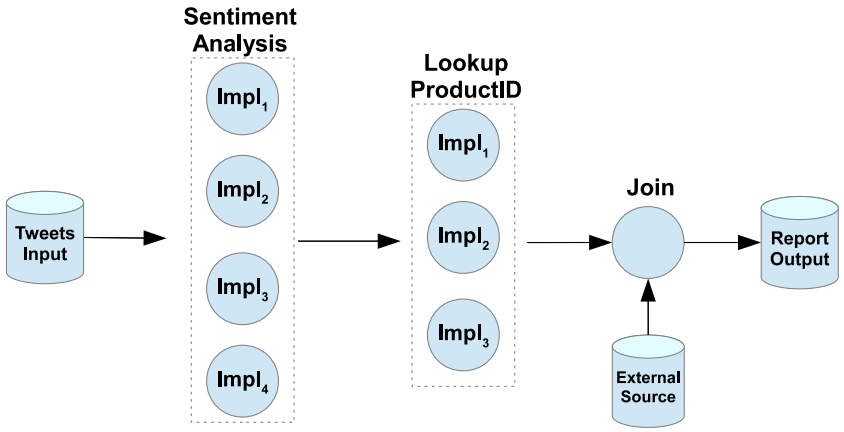

Fig. 8 An example where *Task Implementation Selection* is applicable, where there are four equivalent ways to implement sentiment analysis and three ways to extract product ids

multiple candidates in a setting where the workflows mainly consist of a series of remote WS invocations appears in [22], which is further extended by Tao et al. [96].

Based on the above descriptions, two main observations can be drawn regarding the majority of the techniques. Firstly, they address a multi-objective problem. Secondly, they are proposed for a WS application domain. The latter may imply that transferring the results to data flows where tasks exchange big volumes of data directly may not be straightforward. As a final note, there are numerous proposals that perform task implementation selection considering specific types of tasks, such as classification tasks in data mining data flows (e.g., [73]), and file descriptors in ETLs (e.g., [79]). We do not discuss in detail such techniques, because they do not meet the criteria in Sect. 2.2; further, when generalized to arbitrary tasks, they typically correspond to non-interesting enumeration solutions.

4.7 Execution engine selection

The techniques in this category focus on choosing the best execution engine for executing the data flow tasks in distributed environments, where there are multiple options. For example, assume that the sentiment analysis in our running example can take place on either a DBMS server or a MapReduce cluster. As previously, for the techniques using this mechanism, the vertex cost of each task for each candidate execution engine is a necessary piece of metadata for the optimization algorithm. Also, corresponding techniques are orthogonal to optimizations referring to the high-level execution plan aspects.

For those tasks that can be executed by multiple engines, an exhaustive solution can be adopted for optimally allocating the tasks of a flow to different execution engines in order to meet multiple objectives. The drawback is that an exhaustive solution in general does not scale for large number of flow tasks and execution engines similarly to the case of task implementation selection. To overcome this, a set of heuristics can be used for pruning the search space [90,91,93]. This technique aims to improve not only the performance, but also the reliability of ETL workflows in terms of fault tolerance. Additionally, a multi-objective solution for optimizing the monetary cost and the performance is to check all the possible execution plans that satisfy a specific time constraint; this approach cannot scale for execution plans with high number of operators. The objective functions are those mentioned in Sect. 4.1. The same approach to deciding the execution engine can be used to choose the task implementation in [90,91,93].

Anytime single-objective heuristics for choosing between multiple engine have been proposed Kougka et al. [63]. Such heuristics take into account, apart from vertex costs, the edge costs and constraints on the capability of an engine to execute certain tasks and are coupled with a dynamic programming pseudo-polynomial algorithm that can find optimal allocation for a specific form of DAG shapes, namely linear ones. The objective function is minimizing the sum of the costs for both tasks and edges, extending the definition in Table 3: $\min \sum c(v_i, e_{ij})$, where $i, j = 1 \ldots n$. An extension in [36] explains how these techniques can be extended to optimizing the degree of parallelism in Spark flows taking into account two criteria.

A different approach to engine selection has appeared in the commercial tools in [1,48]. There, the main option is ETL operators to execute on a specialized data integration server, unless a heuristic decides to delegate the execution of some of the tasks to the underlying databases, after merging the tasks and reformulating them as a single query.

Finally, the engine selection mechanism can be employed in combination with configuration of execution engine parameters. An example technique is presented by Huang et al. [44], where the initial optimization step deals with the decision of the best type of execution engine and then, the configuration parameters are defined, as it is analyzed in Sect. 4.8. This technique is extended by Huang et al. [46], which focuses on how to decide on the usage of a specific

type of cloud machines, namely spot instances. The problem of deciding whether to employ spot instances in clouds is also considered by Zhou et al. [113].

4.8 Execution engine configuration

This type of flow optimization has recently received attention due to the increasing number of parallel data flow platforms, such as Hadoop and Spark. The *Engine Configuration* mechanism can serve as a complementary component of an optimization technique that applies implementation or engine selection, and in general, can be combined with the other optimization mechanisms. For example, the rationale of the heuristic presented by Kumbhare et al. [67] (based on variable sized bin packing) is also to decide the best implementation for each task and then, dynamically configure the resources, such as the number of CPU cores allocated, for executing the tasks. A common feature of all the solutions in this section is that they deal with parallelism, but from different perspectives depending on the exact execution environment.

A specific type of engine configuration, namely to decide the degree of parallelism in MapReduce-like clusters for each task and parameters, such as the number of slots on each node, appears in [44]. The time complexity of this optimization technique is exponential. This is repeated for each different type of machines (i.e., different type of execution engine), assuming a context where several heterogeneous clusters are at user's disposal. Both of these techniques have been proposed for cloud environments and aim to optimize multiple optimization criteria.

In general, execution engines come with a large number of configuration parameters and fine tuning them is a challenging task. For example, MapReduce systems may have more than one hundred configuration parameters. The proposal in [84] aims to provide a principle approach to their configuration. Given the number of MapReduce slots and hardware details, the proposed algorithm initially checks all combinations of four key parameters, such as the number of map and reduce waves, and whether to use compression or not. Then, the values of a dozen other configuration parameters that have significant impact on performance are derived. The overall goal is to reduce the execution time taking to account the pipeline nature of MapReduce execution.

An alternative configuration technique is employed by Lim et al. [70], which leverages the what-if engine initially proposed by Herodotou et al. [41]. This engine is responsible to configure execution settings, such as memory allocation and number of map and reduce tasks, by answering questions on real and hypothetical input parameters using a random search algorithm. What-if analysis is also employed by Huang et al. [45] for optimally configuring memory configurations. The distinctive feature of this proposal is that it is dynamic in the sense that it can take decisions at runtime leading to task migrations.

In a more traditional ETL setting, apart from the optimizations described previously, an additional optimization mechanism has been proposed by Simitsis et al. [92] in order to define the degree of parallelism. Specifically, due to the large size of data that a workflow has to process, data are partitioned to be executed following the intra-operator parallelism paradigm. The parallelism is considered profitable whenever the overhead of data partitioning and merging does not incur an overhead higher than the expected benefits. Sometimes, it might be worth investigating whether splitting an input dataset into partitions could reduce the latency in ETL flow execution on a single server as well. An example study can be found in [72].

Another approach to choosing the degree of parallelism appears in [57], where a set of greedy and simulated annealing heuristics that decide the degree of parallelism are proposed. This proposal considers two objectives, performance and monetary cost assuming that resources are offered by a public cloud at a certain price. The objective function targets either the minimization of the sum of the task costs constrained by a defined monetary budget, or the minimization of the monetary cost under a constraint on runtime. Additionally, both metrics can be minimized simultaneously using an appropriate objective function, which expresses the speedup when budget is increased.

Another optimization technique in [25] proposes a set of optimizations at the chip processor level and more specifically, proposes heuristics to drive compiler decisions on whether to execute low-level commands in a pipelined fashion or to employ SIMD (single instruction multiple data) parallelism. Interestingly, these optimizations are coupled with traditional database-like ones at a higher level, such as pushing selections as early as possible.

5 Evaluation approaches

The purpose of this section is to describe what approach the authors of the proposals have followed to evaluate their work. Due to the diversity of the objectives and the lack of a common and comprehensive evaluation approach and benchmark, in general, the proposals are not comparable to each other; therefore, no performance evaluation results are presented.

We can divide the proposals in three categories (see also Fig. 9). The first category includes the optimization proposals that are theoretical in their nature and their results are not accompanied by experiments. Examples of this category are [6,31]. The second category consists of optimizations that have found their way into data flow tools; the only examples in this category are [1,48].

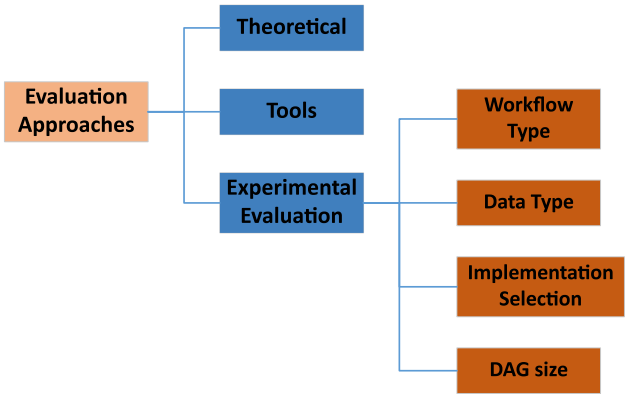

Fig. 9 The three main evaluation approaches followed and the aspects discussed in the experimental one

The third category covers the majority of the proposals, for which experimental evaluation has been provided. We are mostly interested in three aspects of such experiments, namely the *workflow type* used in the experiments, the *data type* used to instantiate the workflows, and the *implementation environment* of the experiments. In Table 4, the experimental evaluation approaches are summarized, along with the maximum DAG size (in terms of number of tasks) employed. Specifically, the implementation environment defines the execution environment of a workflow during the evaluation procedure. The environment can be a *real-world* one, which considers either the customization of an existing system to support the proposed optimization solutions or the design of a prototype system, which is essentially a new platform, possibly designed from scratch and tailored to support the evaluation. A common approach consists of a *simulation* of a real execution environment. Discussing the pros and cons of each approach is out of our scope, but in general, simulations allow the experimentation with a broader range of flow types, whereas real experiments can better reveal the actual benefits of optimizations in practice.

The type of the workflows considered are either synthetic or real-world. In the former case, arbitrary DAGs are produced, e.g., based on the guidelines in [86]. In the latter case, the flow structure is according to real-world cases. For example, the evaluation of [22,24,30,57,63,113] is based on real-world scientific workflows, such as the Montage and Cybershake ones described in [55]. Another example of real-world workflows are derived by TPC-H queries (used for some of the evaluation experiments in [47,70,81] along with real-world text mining and information extraction examples). In [90–93], the evaluation of the optimization proposals is based on workflows that represent arbitrary, real-world data transformations and text analytics. The case studies in [25,70] include standard analytical algorithms, such as PageRank, k-means, logistic regression, and naive bayes.

The datasets used for workflow execution may affect the evaluation results, since they specify the range of the statistical metadata considered. The processed datasets can be either synthetic or real ones extracted by repositories, such as the Twitter repository with sample data of real tweets. Examples of real datasets used in [47,81] include biomedical texts, a set of Wikipedia articles, and datasets from DBpedia. Additionally, Braga et al. [15] have evaluated the proposed optimization techniques using real data extracted by www.conference-service.com, www.bookings.com, and www.accuweather.com. Typically, when employing standard scientific flows, the datasets used are also fixed; however, in [63] a wide range of artificially created metadata have been used to cover more cases.

As shown in Table 4, a big portion of the optimization techniques have been evaluated by executing workflows in a simulated environment. The real environments that have been employed include among others ETL tools, such as Kettle and Talend, extensions to MapReduce, tailored prototypes, and DBMSs.

Finally, for many techniques, only small data flows comprising no more than 15 nodes were used, or the information with regard to the size of the flows could not be derived. In the latter case, this might be due to the fact that well-known algorithms have been used (e.g., k-means in [25] and matrix-multiplication in [44]) without explaining how these algorithms are internally translated to data flows. All experiments with workflows comprising hundreds of tasks used synthetic datasets.

6 Discussion on findings

Data flow optimization is a research area with high potential for further improvements given the increasing role of data flows in modern data-driven applications. In this survey, we have listed more than thirty research proposals, most of which have been published after 2010. In the previous sections, we mostly focused on the merits and the technical details of each proposal. They can lead to performance improvements, and more importantly, they have the potential to lift the burden of manually fixing all implementation details from the data flow designers, which is a key motivation for automated optimization solutions. In this section, we complement any remarks made before with a list of additional observations, which may also serve as a description of directions for further research:

– In principle, the techniques described previously can serve as building block toward more holistic solutions. For instance, task ordering can, in principle, be combined with (i) additional high-level mechanisms, such as task introduction, removal, merge, and decomposition; and (ii) low-level mechanisms, such as engine configura-

Table 4 Experimental evaluation of proposals

(References, years)	Workflow type	Data type	Implementation environment	Max. DAG size
([40], 1998)	Synthetic	Synthetic	Real (DBMS)	4
([20], 1999)	Synthetic	Synthetic	Real (DBMS)	16
([111], 1999)	Synthetic	Synthetic	Simulation	15
([87], 2005)	Synthetic	Synthetic	Simulation	70
([30], 2005)	Real	Real	Real (Pegasus)	N/A
([108], 2005)	Synthetic	Synthetic	Simulation	200
([94], 2006)	Synthetic	Synthetic	Real (ad hoc prototype)	4
([101], 2007)	Synthetic	Synthetic	Simulation	15
([107], 2007)	Synthetic	Synthetic	Real (Web-Sphere Process Server [107])	N/A
([15], 2008)	Real	Real	Simulation	7
([68], 2008)	Synthetic	Synthetic	Real (ad hoc prototype)	8
([22], 2009)	Synthetic	Synthetic	Simulation	120
([65], 2010)	Synthetic	Synthetic	Simulation	60
([92], 2010)	Real	Synthetic	Real (Kettle ETL tool)	80
([57], 2011)	Real	Synthetic	Real (ADP prototype)	500
([98,99], 2011)	Synthetic	Synthetic	Simulation	100
([47], 2012), ([81], 2015)	Real	Real	Real (Stratosphere [8])	15
([70], 2012)	Real	Synthetic	Real (extensions to MapReduce)	14
([90], 2012), ([91,93], 2013)	Real	Real	Real (Kettle ETL tool)	15
([44], 2013), ([46], 2015)	Real	Synthetic	Real (extensions to MapReduce)	N/A
([67], 2013)	Synthetic	Synthetic	Simulation	4
([63], 2014)	Real	Synthetic	Simulation	200
([84], 2014)	Real	Synthetic	Real (extensions to MapReduce)	< 10
([24], 2014)	Real	Real	Real (Taverna)	N/A
([25], 2015)	Real	Synthetic	Real (Tupleware prototype)	N/A
([45], 2015)	Real	Real	Real (extensions to MapReduce)	N/A
([60], 2015), ([59], 2014)	Synthetic	Synthetic	Simulation	200
([72], 2015)	Real	Synthetic	Real (Talend ETL tool)	11
([113], 2015)	Real	Synthetic	Both (Pegasus)	$> 10{,}000$

tion, thus yielding added benefits. The main issue arising when mechanisms are combined is the increased complexity. An approach to mitigating the complexity is a two-phase approach, as commonly happens in database queries. An additional issue is to determine which mechanism should first be explored. For some mechanisms, this is straightforward, e.g., decomposition should precede task ordering and task removal should be placed afterward. But, for mechanisms, such as configuration, this is unclear, e.g., whether it is beneficial to first configure low-level details before higher level ones remains an open issue.

– In general, there is little work on low-complexity, holistic, and multi-objective solutions. Toward this direction, Simitsis et al. [92] consider more than one objective and combines mechanisms at both high- and low-level execution plan details; for instance, both task ordering and engine configuration are addressed in the same technique. But clearly more work is needed here. In general, most of the techniques have been developed in isolation, each one typically assuming a specific setting and targeting a subset of optimization aspects. This and the lack of a common agreed benchmark makes it difficult to understand how exactly they compare to each other, the details of how the various proposals can be combined in a common framework and how they interplay.
– There seems to be no common approach to evaluating the optimization proposals. Some proposals have not been adequately tested in terms of scalability, since they have considered only small graphs. In some data flow evaluations, workloads inspired from benchmarks such as TPC-DI/DS have been employed, but as most of the authors report as well, it is doubtful whether these benchmarks can completely capture all dimensions of the

problem. There is a growing need for the development of systematic and broadly adopted techniques to evaluate optimization techniques for data flows.
- A significant part of the techniques covered in this survey have not been incorporated in tools, nor have been exploited commercially. Most of the optimization techniques described here, especially regarding the high-level execution plan details, have not been implemented in real data flow systems apart from very few exceptions, as explained earlier. Hence, the full potential and practical value of the proposals have not been investigated in actual execution conditions, despite the fact that evaluation results thus far are shown to provide improvements by several orders of magnitude over non-optimized plans.
- A plethora of objective functions and cost models have been investigated, which, to a large extent, they are compatible with each other, despite the fact that original proposals have examined them in isolation. However, it is unclear whether any of such cost models can capture aspects, such as the execution time of parallel data flows, which are very common nowadays, in a fairly accurate manner. A more sophisticated cost model should take into account sequential, pipelined and partitioned execution in a unified manner, essentially combining the sum, bottleneck and critical path cost metrics. An early work on this topic has appeared in [62].
- Developing adaptive solutions that are capable of revising the flow execution plan on the fly is one important open issue, especially for online, continuous, and stream processing. Also, very few optimization techniques consider the cost of the graph edges. Not considering edge metadata does not reflect entirely real data flow execution in distributed settings, where the cost of transmitting data depends both on sender and receiver.
- In this survey, we investigated single flow optimizations. Optimizing multiple flows simultaneously is another area requiring attention. An initial effort is described by Jovanovic et al. [52], which builds upon the task ordering solutions of [87].
- There is early work on statistics collection [23,39,76,85], but clearly, there is more to be done here given that without appropriate statistics, cost-based optimization becomes problematic and prone to significant errors.
- On the other hand, a different school of thought advocates that in contrast to relational databases, automated optimization cannot help in practice in flow optimization due to flow complexity and increased difficulty in maintaining flow statistics, and developing accurate cost models. Based on that, there is a number of commercial flow execution engines (e.g., ETL tools) that instead of offering a flow optimizer they provide users with tips and best practices. No doubt, this is an interesting point, but we consider this category as out of the scope of this work.

6.1 Future research directions

Given the above observations and the trend in developing new solutions in the recent years, data flow optimization seems to be technology in evolution rather than an area, where most significant problems have been resolved. Moreover, providing solutions to all these problems is more likely to yield significantly different and more powerful new approaches to data flow optimization, rather than delta improvements on existing solutions.

The main future research directions foreseen in this survey directly relate to tackling the limitations implied by the observations above, and call for a paradigm shift toward:

- *Multiple optimization mechanisms considered concurrently*. The fact that data flows increasingly operate on continuously arriving and evolving data renders task ordering a key optimization mechanism. But since modern data flow engines provide multiple alternatives ranging from the implementation type to the degree of parallelism and geo-distributed data analytics is becoming a reality, task ordering needs to be combined with all lower-level mechanisms defined. This will further explode the already exponential search space. As mentioned above, two-phase optimization solutions are a promising approach to tackle scalability issues, but may significantly diverge from good solutions because the optimizations on one level directly impact on decisions on the other, e.g., re-ordering tasks may ungroup tasks supposed to run a single location.
- *Multiple and additional KPIs (key performance indicators)*. Novel data flow optimization solutions are foreseen to account for at least two objectives, for example, running time and monetary costs for employing cloud resources. This entails that both bi-objective techniques need to be developed and the corresponding cost models need to be devised.
- *Several flows optimized simultaneously.* Modern data flow engines very commonly run on top of clusters, which already benefit from managers, such as YARN and MESOS, that take responsibility for sharing resources among multiple users and applications. Since the execution layer naturally supports simultaneous flow executions, a step going beyond the current state-of-the-art in data flow optimization is to account for such concurrent flows.
- *End-to-end optimization solutions*. The optimization solutions cannot be incorporated into real systems unless the practical issues of acquiring and maintaining flow statistics are resolved; therefore, data flow metadata management is a promising direction for future research.

7 Additional issues in data-centric flow optimization;

Additional issues are split into four parts. First, we describe optimizations enabled in current state-of-the-art parallel data flow systems, which, however, cannot cover arbitrary DAGs and tasks, and as such, have not been included in the previous sections. Next, we discuss techniques that although they do not perform optimization in their own, they could, in principle, facilitate optimization. We provide a brief overview of optimization solutions for the WEP execution layer, complementing the discussion of existing scheduling techniques in Sect. 8. We conclude with a brief note on implementing the optimization techniques into existing systems.

7.1 Optimization in massively parallel data flow systems

A specific form of data flow systems are massively parallel processing (MPP) engines, such as Spark and Hadoop. These data flow systems can scale to a large number of computing nodes and are specifically tailored to big data management taking care of parallelism efficiency and fault tolerance issues. They accept their input in a declarative form (e.g., PigLatin [75], Hive, SparkSQL), which is then automatically transformed into an executable DAG. Several optimizations take place during this transformation.

We broadly classify these optimizations in two categories. The first category comprises database-like optimizations, such as pushing filtering tasks as early as possible, choosing the join implementation, and using index tables, corresponding to *task ordering* and *implementation selection*, respectively. This can be regarded as a direct technology transfer from databases to parallel data flows and to date, these optimizations do not cover arbitrary user-defined transformations.

The second category is specific to the parallel execution environment with a view to minimizing the amount of data read from disk, transmitted over the network, and being processed. For example, Spark groups pipelining tasks in larger jobs (called stages) to benefit from this type of parallelism. Also, it leverages cached data and columnar storage, performs compression, and reduces the amount of data transmitted during data shuffling through early partial aggregation, when this is possible. Grouping tasks into pipelining stages is a case of runtime scheduling. Early partial aggregation can be deemed as a *task introduction* technique. The other forms of optimizations (leveraging cached data, columnar storage, and compression) can be deemed as specific forms of *implementation selection*. Flink is another system employing optimizations, but it has not yet incorporated all the (advanced) optimization proposals in its predecessor projects, as described in [47,81]. The proposal in [14] is another example that proposes optimizations for a specific operator, namely *ParFOR*.

We do not include these techniques in Tables 1 and 2 because they apply to specific DAG instances and have not matured enough to benefit generic data flows including arbitrary tasks. Finally, in terms of scheduling tools for data-intensive flows, several software artefacts have started emerging, such as Apache Oozie, Apache Cascading. We also do not cover these because they refer to the WEP execution rather than the WEP generation layer.

7.2 Techniques facilitating data-centric flow optimization

Statistical metadata, such as cost per task invocation and selectivity, play a significant role in data flow optimization as discussed previously. References [23,39,76,85] deal with statistics collection and modeling the execution cost of workflows; such issues are essential components in performing sophisticated flow optimization. Vassiliadis et al. [104] analyze the properties of tasks, e.g., multiple-input vs single-input ones; such properties along with dependency constraint information complement statistics as the basis on top of which optimization solutions can be built.

In principle, algebraic approaches to workflow execution and modeling facilitate flow optimization, e.g., in establishing dependency constraints. Examples of such proposals appear in [74,81]. The techniques that we discuss go beyond any type of modeling; however, when an algebraic approach is followed, further operator-specific optimizations become possible capitalizing on the vast literature of query optimization as already mentioned.

Some techniques allow for choosing among multiple implementations of the same tasks using ontologies, rather than performing cost-based or heuristic optimization [28]. In [109], improving the flow with the help of user interactions is discussed. Additionally, in [74], different scheduling strategies to account for data shipping between tasks are presented, without however proposing an optimization algorithm that takes decisions as to which strategy should be employed.

Apart from the optimizations described in Sect. 4, the proposal in [92] considers also the objective of data freshness. To this end, the proposal optimizes the activation time of ETL data flows, so that the changes in data sources are reflected on the state of a Data Warehouse within a time window. Nevertheless, this type of optimization objective leads to techniques that do not focus on optimizing the flow execution plan per se, which is the main topic of this survey.

For the evaluation of optimization proposals, benchmarks for evaluating techniques are proposed in [86,88]. Finally, in [42,66], the significant role of correct parameter configuration in large-scale workflow execution is identified and relevant approaches are proposed. Proper tuning of the data

flow execution environment is orthogonal and complementary to optimization of flow execution plan.

7.3 On scheduling optimizations in data-centric flows

In general, data flow execution engines tend to have built-in scheduling policies, which are not configured on a single flow basis. In principle, such policies can be extended to take into account the specific characteristics of data flows, where the placement of data and the transmission of data across tasks, represented by the DAG edges, requires special attention [21]. For example, in [56], a set of scheduling strategies for improving the performance through the minimization of memory consumption and the execution time of Extract–Transform–Load (ETL) workflows running on a single machine is proposed. As it is difficult to execute the data in pipeline in ETLs due to the blocking nature of some of the ETL tasks, the authors suggest splitting the workflow into several sub-flows and apply different scheduling policies if necessary. Finally, in [50], the placement of data management tasks is decided according to the memory availability of resources taking into account the trade-off between co-locating tasks and the increased memory consumption when running multiple tasks on the same physical computational node.

A large set of scheduling proposals target specific execution environments. For example, the technique in [38] targets shared resource environments. Proposals, such as [17,22,66,80,82,112], are specific to grid and cloud data-centric flow scheduling. Agrawal et al. [7] discuss optimal time schedules given a fixed allocation of tasks to engines, provided that the tasks belong to a linear workflow.

Also, a set of optimization algorithms for scheduling flows based on deadline and time constraints is analyzed in [3,4]. Another proposal of flow scheduling optimization is presented in [77] based on soft deadline rescheduling in order to deal with the problem of fault tolerance in flow executions. In [17], an optimization technique for minimizing the performance fluctuations that might occur by the resource diversity, which also considers deadlines, is proposed. Additionally, there is a set of scheduling techniques based on multi-objective optimization, e.g., [33].

7.4 On incorporation optimization techniques into existing systems

Without loss of generality, there are two main types of describing the data flow execution plan in existing tools and prototypes: either in an appropriately formatted text file or using internal representations in the code. These two approaches are exemplified in systems, like the Pentaho Kettle, Spark, Taverna, and numerous others. In the former case, an optimization technique can be inserted as a component that processes this text file and produces a different execution plan. As an example, in Pentaho, each task and each graph edge are described as different XML elements in an XML document. Then, a technique that performs task reordering can consist of an independent programming module that parses the XML file and modifies the edge elements. On the other hand, systems, such as Spark, transform the flow submitted by the user in a DAG, but without exposing a high-level representation to the end user. The internal optimization component, called Catalyst, then performs modifications to the internal code structure that captures the executable DAG. Extending the optimizer to add new techniques, such as those described in this survey, requires using the Catalyst extensibility points. The second approach seems to require more effort from the developer and be more intrusive. Finally, tools that allow for rapid feedback to the developer and the human expert designer being in the loop, e.g., as in [43], can also benefit for automated optimization solutions like those discussed in this survey.

8 Related work

To the best of our knowledge, there is no prior survey or overview article on data flow optimization; however, there are several surveys on related topics.

Related work falls into two categories: (i) surveys on generic DAG scheduling and on narrow-scope scheduling problems, which are also encountered in data flow optimization; and (ii) overviews of workflow systems.

DAG scheduling is a persisting topic in computing and has received a renewed attention due to the emergence of Grid and cloud infrastructures, which allow for the usage of remote computational resources. For such distributed settings, the proposals tend to refer to the WEP execution layer and to focus on mapping computational tasks ignoring the data transfer between them, or assume a non-pipelined mode of execution that does not fit will into data-centric flow setting [32]. A more recent survey of task mapping is presented in [37], which discusses techniques that assign tasks to resources for efficient execution in Grids under the demanding requirements and resource allocation constraints, such as the dependencies between the tasks, the resource reservation, and so on. In [10], an overview of the pipelined workflow time scheduling problem is presented, where the problem formulation targets streaming applications. In order to compare the effectiveness of the proposed optimization techniques, they present a taxonomy of workflow optimization techniques taking into account workflow characteristics, such as the structure of flow (i.e., linear, fork, tree-shaped DAGs), the computation requirements, the size of data to be transferred between tasks, the parallel or sequential task exe-

cution mode, and the possibility of executing task replicas. Additionally, the taxonomy takes into consideration a performance model that describes whether the optimization aims to a single or multiple objectives, such as throughput, latency, reliability, and so on. However, in data-centric flows, tasks are activated upon receipt of input data and not as a result of an activation message from a controller, as assumed in [10]. None of the surveys above provides a systematic study of the optimizations at the WEP generation layer.

The second class of related work deals with a broader-scope presentation of workflow systems. The survey in [29] aims to present a taxonomy of the workflow system features and capabilities to allow end users to take the best option for each application. Specifically, the taxonomy is inspired by the workflow lifecycle and categorizes the workflow systems according to the lifecycle phase they are capable of supporting. However, the optimizations considered suffer from the same limitations as those in [32]. Similarly, in [9], an evaluation of the current workflow technology is also described, considering both scientific and business workflow frameworks. The control and data flow mechanisms and capabilities of workflow systems both for e-science, e.g., Taverna and Triana, and business processes, e.g., YAWL and BPEL-based engines, are discussed in [26]. [106] discusses how leading commercial tools in the data analysis market handle SQL statements, as a means to perform data management tasks within workflows. Liu et al. [71] focus on scientific workflows, which are an essential part of data flows, but does not delve into the details of optimization. Finally, Jovanovic et al. [51] present a survey that aims to present the challenges of modern data flows through different data flow scenarios. Additionally, related data flow optimization techniques are summarized, but not surveyed, in order to underline the importance of low data latency in Business Intelligence (BI) processes, while an architecture of next generation BI systems that manage the complexity of modern data flows in such systems is proposed.

Modeling and processing ETL workflows [103] focuses on the detailed description of conceptual and logical modeling of ETLs. Conceptual modeling refers to the initial design of ETL processes by using UML diagrams, while the logical modeling refers to the design of ETL processes taking into account required constraints. This survey discusses the generic problems in ETL data flows, including optimization issues in minimizing the execution time of an ETL workflow and the resumption in case of failures during the processing of large amount of data.

Data flow optimization bears also similarities with query optimization over Web Services (WSs) [100], especially when the valid orderings of the calls to the WSs are subject to dependency constraints. This survey includes all the WSs related techniques that can also be applied to data flows.

Part of the optimizations covered in this survey can be deemed as generalizations of the corresponding techniques in database queries. An example is the correspondence between pushing selections down in the query plan and moving filtering tasks as close to data source as possible [12]. Comprehensive surveys on database query optimization are in [18,49], whereas lists of semantic equivalence rules between expressions of relational operators that provide the basis for query optimization can be found in classical database textbooks (e.g., [35]). However, as discussed in the introduction, there are essential differences between database queries and data flows, which cannot be described as expressions over a limited set of elementary operations. At a higher level, data flow optimization covers more mechanisms (e.g., task decomposition and engine selection) and a broader setting with regard to the criteria considered and the metadata required.

Nevertheless, it is arguable that data flow task ordering bears similarities to optimization of database queries containing user-defined functions (UDFs) (or expensive predicates), as reported in [20,40]. This similarity is based on the intrinsic correspondence between UDFs and data flow tasks, but there are two main differences. First, the dependency constraints considered in [20,40] refer to pairs of a join and a UDF, rather than between UDFs. As such, when joins are removed and only UDFs are considered, the techniques described in these proposals are reduced to unconstrained filter ordering. Second, the straightforward extensions to the proposals [20,40] are already covered and improved by solutions targeting data flow task ordering explicitly as discussed in Sect. 4.1.

9 Summary

This survey covers an emerging area in data management, namely optimization techniques that modify a data-centric workflow execution plan prior to its execution in an automated manner. The survey first provides a taxonomy of the main dimensions characterizing each optimization proposal. These dimensions cover a broad range, from the mechanism utilized to enhance execution plans to the distribution of the setting and the environment for which the solution is initially proposed. Then, we present the details of the existing proposals, divided into eight groups, one for each of the identified optimization mechanisms. Next, we present the evaluation approaches, focusing on aspects, such as the type of workflows and data used during experiments. We complete this survey with a discussion of the main findings, while also, for completeness, we briefly present tangential issues, such as optimizations in massively parallel data flow systems and optimized workflow scheduling.

References

1. IBM infosphere datastage balanced optimization. http://www-01.ibm.com/software/data/integration/info_server/ (2008). Accessed Jan 2018
2. Abadi, D.J., Agrawal, R., Ailamaki, A., Balazinska, M., Bernstein, P.A., Carey, M.J., Chaudhuri, S., Dean, J., Doan, A., Franklin, M.J., Gehrke, J., Haas, L.M., Halevy, A.Y., Hellerstein, J.M., Ioannidis, Y.E., Jagadish, H.V., Kossmann, D., Madden, S., Mehrotra, S., Milo, T., Naughton, J.F., Ramakrishnan, R., Markl, V., Olston, C., Ooi, B.C., Ré, C., Suciu, D., Stonebraker, M., Walter, T., Widom, J.: The beckman report on database research. SIGMOD Rec. **43**(3), 61–70 (2014)
3. Abrishami, S., Naghibzadeh, M., Epema, D.H.: Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Future Gener. Comput. Syst. **29**(1), 158–169 (2013)
4. Abrishami, S., Naghibzadeh, M., Epema, D.H.J.: Cost-driven scheduling of grid workflows using partial critical paths. IEEE Trans. Parallel Distrib. Syst. **23**(8), 1400–1414 (2012)
5. Agrawal, K., Benoit, A., Dufossé, F., Robert, Y.: Mapping filtering streaming applications with communication costs. In: SPAA, pp. 19–28 (2009)
6. Agrawal, K., Benoit, A., Dufossé, F., Robert, Y.: Mapping filtering streaming applications. Algorithmica **62**(1–2), 258–308 (2012)
7. Agrawal, K., Benoit, A., Magnan, L., Robert, Y.: Scheduling algorithms for linear workflow optimization. In: IPDPS, pp. 1–12 (2010)
8. Alexandrov, A., Bergmann, R., Ewen, S., Freytag, J., Hueske, F., Heise, A., Kao, O., Leich, M., Leser, U., Markl, V., Naumann, F., Peters, M., Rheinländer, A., Sax, M.J., Schelter, S., Höger, M., Tzoumas, K., Warneke, D.: The stratosphere platform for big data analytics. VLDB J. **23**(6), 939–964 (2014)
9. Barker, A., van Hemert, J.I.: Scientific workflow: a survey and research directions. In: PPAM, Lecture Notes in Computer Science, vol. 4967, pp. 746–753 (2007)
10. Benoit, A., Çatalyürek, U.V., Robert, Y., Saule, E.: A survey of pipelined workflow scheduling: models and algorithms. ACM Comput. Surv. **45**(4), 50:1–50:36 (2013)
11. Bhattacharya, K., Hull, R., Su, J.: A data-centric design methodology for business processes. In: Handbook of Research on Business Process Modeling, Chapter 23, 503–531 (2009)
12. Böhm, M.: Cost-based optimization of integration flows. Ph.D. thesis (2011)
13. Böhm, M., Habich, D., Lehner, W.: On-demand re-optimization of integration flows. Inf. Syst. **45**, 1–17 (2014)
14. Böhm, M., Tatikonda, S., Reinwald, B., Sen, P., Tian, Y., Burdick, D., Vaithyanathan, S.: Hybrid parallelization strategies for large-scale machine learning in systemml. PVLDB **7**(7), 553–564 (2014)
15. Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Optimization of multi-domain queries on the web. PVLDB **1**(1), 562–573 (2008)
16. Burge, J., Munagala, K., Srivastava, U.: Ordering pipelined query operators with precedence constraints. Technical Report 2005-40, Stanford InfoLab (2005)
17. Calheiros, R.N., Buyya, R.: Meeting deadlines of scientific workflows in public clouds with tasks replication. IEEE Trans. Parallel Distrib. Syst. **25**(7), 1787–1796 (2014)
18. Chaudhuri, S.: An overview of query optimization in relational systems. In: Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1–3, 1998, Seattle, Washington, pp. 34–43 (1998)
19. Chaudhuri, S., Dayal, U., Narasayya, V.: An overview of business intelligence technology. Commun. ACM **54**, 88–98 (2011)
20. Chaudhuri, S., Shim, K.: Optimization of queries with user-defined predicates. ACM Trans. Database Syst. **24**(2), 177–228 (1999)
21. Chen, W., Deelman, E.: Partitioning and scheduling workflows across multiple sites with storage constraints. In: Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics—Volume Part II, PPAM'11, pp. 11–20 (2012)
22. Chen, W.N., Zhang, J.: An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **39**(1), 29–43 (2009)
23. Chirkin, A.M., Belloum, A., Kovalchuk, S.V., Makkes, M.X.: Execution time estimation for workflow scheduling. In: Proceedings of the 9th Workshop on Workflows in Support of Large-Scale Science, pp. 1–10. IEEE Press (2014)
24. Cohen-Boulakia, S., Chen, J., Goble, C., Missier, P., Williams, A., Froidevaux, C.: Distilling structure in taverna scientific workflows: a refactoring approach. BMC Bioinformatics **15**(1), S12 (2014)
25. Crotty, A., Galakatos, A., Dursun, K., Kraska, T., Binnig, C., Çetintemel, U., Zdonik, S.: An architecture for compiling udf-centric workflows. PVLDB **8**(12), 1466–1477 (2015)
26. Curcin, V., Ghanem, M.: Scientific workflow systems—can one size fit all? In: Biomedical Engineering Conference, 2008. CIBEC 2008. Cairo International, pp. 1–9 (2008)
27. Dayal, U., Castellanos, M., Simitsis, A., Wilkinson, K.: Data integration flows for business intelligence. In: Proceedings of EDBT, pp. 1–11 (2009)
28. de Oliveira, D., Ogasawara, E.S., Dias, J., Baio, F.A., Mattoso, M.: Ontology-based semi-automatic workflow composition. JIDM **3**(1), 61–72 (2012)
29. Deelman, E., Gannon, D., Shields, M., Taylor, I.: Workflows and e-science: an overview of workflow system features and capabilities. Future Gener. Comput. Syst. **25**(5), 528–540 (2009)
30. Deelman, E., Singh, G., Su, M.H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A., Jacob, J.C., Katz, D.S.: Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci. Program. **13**(3), 219–237 (2005)
31. Deshpande, A., Hellerstein, L.: Parallel pipelined filter ordering with precedence constraints. ACM Trans. Algorithms **8**(4), 41:1–41:38 (2012)
32. Dong, F., Akl, S.G.: Scheduling algorithms for grid computing: state of the art and open problems. Technical report (2006)
33. Fard, H., Prodan, R., Fahringer, T.: A truthful dynamic workflow scheduling mechanism for commercial multicloud environments. IEEE Trans. Parallel Distrib. Syst. **24**(6), 1203–1212 (2013)
34. Florescu, D., Levy, A., Manolescu, I., Suciu, D.: Query optimization in the presence of limited access patterns. In: ACM SIGMOD, pp. 311–322 (1999)
35. Garcia-Molina, H., Ullman, J.D., Widom, J.D.: Database Systems: The Complete Book. Prentice Hall, Upper Saddle River (2001)
36. Gounaris, A., Kougka, G., Tous, R., Tripiana, C., Torres, J.: Dynamic configuration of partitioning in spark applications. IEEE Trans. Parallel Distrib. Syst. (2017). https://doi.org/10.1109/TPDS.2017.2647939
37. Grehant, X., Demeure, I., Jarp, S.: A survey of task mapping on production grids. ACM Comput. Surv. **45**(3), 37:1–37:25 (2013)
38. Gu, Y., Wu, Q., Rao, N.S.V.: Analyzing execution dynamics of scientific workflows for latency minimization in resource sharing environments. In: Proceedings of the 2011 IEEE World Congress on Services, pp. 153–160 (2011)

39. Halasipuram, R., Deshpande, P.M., Padmanabhan, S.: Determining essential statistics for cost based optimization of an ETL workflow. In: EDBT, pp. 307–318 (2014)
40. Hellerstein, J.M.: Optimization techniques for queries with expensive methods. ACM Trans. Database Syst. **23**(2), 113–157 (1998)
41. Herodotou, H., Babu, S.: Profiling, what-if analysis, and cost-based optimization of mapreduce programs. PVLDB **4**(11), 1111–1122 (2011)
42. Holl, S., Zimmermann, O., Hofmann-Apitius, M.: A new optimization phase for scientific workflow management systems. In: eScience, pp. 1–8 (2012)
43. Holzinger, A., Stocker, C., Ofner, B., Prohaska, G., Brabenetz, A., Hofmann-Wellenhof, R.: Combining HCI, natural language processing, and knowledge discovery—potential of IBM content analytics as an assistive technology in the biomedical field. In: Human–Computer Interaction and Knowledge Discovery in Complex, Unstructured, Big Data—Third International Workshop, HCI-KDD, pp. 13–24 (2013)
44. Huang, B., Babu, S., Yang, J.: Cumulon: optimizing statistical data analysis in the cloud. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 1–12 (2013)
45. Huang, B., Böhm, M., Tian, Y., Reinwald, B., Tatikonda, S., Reiss, F.R.: Resource elasticity for large-scale machine learning. In: SIGMOD'15, pp. 137–152 (2015)
46. Huang, B., Jarrett, N.W.D., Babu, S., Mukherjee, S., Yang, J.: Cümülön: Matrix-based data analytics in the cloud with spot instances. Proc. VLDB Endow. **9**(3), 156–167 (2015)
47. Hueske, F., Peters, M., Sax, M., Rheinländer, A., Bergmann, R., Krettek, A., Tzoumas, K.: Opening the black boxes in data flow optimization. PVLDB **5**(11), 1256–1267 (2012)
48. Informatica: How to achieve flexible, cost-effective scalability and performance through pushdown processing. White Paper (2007)
49. Ioannidis, Y.E.: Query optimization. ACM Comput. Surv. **28**(1), 121–123 (1996)
50. Jin, T., Zhang, F., Sun, Q., Bui, H., Parashar, M., Yu, H., Klasky, S., Podhorszki, N., Abbasi, H.: Using cross-layer adaptations for dynamic data management in large scale coupled scientific workflows. In: International Conference for High Performance Computing, Networking, Storage and Analysis, SC'13, p. 74 (2013)
51. Jovanovic, P., Romero, O., Abelló, A.: A unified view of data-intensive flows in business intelligence systems: a survey. In: Transactions on Large-Scale Data- and Knowledge-Centered Systems XXIX, pp. 66–107. Springer, Berlin (2016)
52. Jovanovic, P., Romero, O., Simitsis, A., Abell, A.: Incremental consolidation of data-intensive multi-flows. IEEE Trans. Knowl. Data Eng. **28**(5), 1203–1216 (2016)
53. Jovanovic, P., Simitsis, A., Wilkinson, K.: Babbleflow: a translator for analytic data flow programs. In: SIGMOD, pp. 713–716 (2014)
54. Jovanovic, P., Simitsis, A., Wilkinson, K.: Engine independence for logical analytic flows. In: ICDE, pp. 1060–1071 (2014)
55. Juve, G., Chervenak, A.L., Deelman, E., Bharathi, S., Mehta, G., Vahi, K.: Characterizing and profiling scientific workflows. Future Gener. Comput. Syst. **29**(3), 682–692 (2013)
56. Karagiannis, A., Vassiliadis, P., Simitsis, A.: Scheduling strategies for efficient ETL execution. Inf. Syst. **38**(6), 927–945 (2013)
57. Kllapi, H., Sitaridi, E., Tsangaris, M.M., Ioannidis, Y.: Schedule optimization for data processing flows on the cloud. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, pp. 289–300 (2011)
58. Kougka, G., Gounaris, A.: Declarative expression and optimization of data-intensive flows. In: DaWaK, pp. 13–25 (2013)
59. Kougka, G., Gounaris, A.: Optimization of data-intensive flows: is it needed? is it solved? In: Proceedings of the 17th International Workshop on Data Warehousing and OLAP, DOLAP 2014, Shanghai, November 3–7, 2014, pp. 95–98 (2014)
60. Kougka, G., Gounaris, A.: Cost optimization of data flows based on task re-ordering. In: LNCS Transactions on Large-Scale Data- and Knowledge-Centered Systems (2017, to appear)
61. Kougka, G., Gounaris, A.: Optimal task ordering in chain data flows: exploring the practicality of non-scalable solutions. In: DaWaK (2017)
62. Kougka, G., Gounaris, A., Leser, U.: Modeling data flow execution in a parallel environment. In: DaWaK (2017)
63. Kougka, G., Gounaris, A., Tsichlas, K.: Practical algorithms for execution engine selection in data flows. Future Gener. Comput. Syst. **45**, 133–148 (2015)
64. Krishnamurthy, R., Boral, H., Zaniolo, C.: Optimization of non-recursive queries. In: VLDB, pp. 128–137 (1986)
65. Kumar, N., Kumar, P.S.: An efficient heuristic for logical optimization of ETL workflows. In: BIRTE, pp. 68–83 (2010)
66. Kumar, V.S., Sadayappan, P., Mehta, G., Vahi, K., Deelman, E., Ratnakar, V., Kim, J., Gil, Y., Hall, M., Kurc, T., Saltz, J.: An integrated framework for parameter-based optimization of scientific workflows. In: HPDC, pp. 177–186 (2009)
67. Kumbhare, A.G., Simmhan, Y., Prasanna, V.K.: Exploiting application dynamism and cloud elasticity for continuous dataflows. In: SC, p. 57 (2013)
68. Kyriazis, D., Tserpes, K., Menychtas, A., Litke, A., Varvarigou, T.A.: An innovative workflow mapping mechanism for grids in the frame of quality of service. Future Gener. Comput. Syst. **24**(6), 498–511 (2008)
69. Li, C.: Computing complete answers to queries in the presence of limited access patterns. VLDB J. **12**(3), 211–227 (2003)
70. Lim, H., Herodotou, H., Babu, S.: Stubby: a transformation-based optimizer for mapreduce workflows. Proc. VLDB Endow. **5**(11), 1196–1207 (2012)
71. Liu, J., Pacitti, E., Valduriez, P., Mattoso, M.: A survey of data-intensive scientific workflow management. J. Grid Comput. **13**(4), 457–493 (2015)
72. Liu, X., Iftikhar, N.: An ETL optimization framework using partitioning and parallelization. In: SAC'15 (2015)
73. Nguyen, P., Hilario, M., Kalousis, A.: Using meta-mining to support data mining workflow planning and optimization. J. Artif. Intell. Res. **51**, 605–644 (2014)
74. Ogasawara, E.S., de Oliveira, D., Valduriez, P., Dias, J., Porto, F., Mattoso, M.: An algebraic approach for data-centric scientific workflows. PVLDB **4**(12), 1328–1339 (2011)
75. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig latin: a not-so-foreign language for data processing. In: SIGMOD Conference, pp. 1099–1110 (2008)
76. Pietri, I., Juve, G., Deelman, E., Sakellariou, R.: A performance model to estimate execution time of scientific workflows on the cloud. In: Proceedings of the 9th Workshop on Workflows in Support of Large-Scale Science, pp. 11–19. IEEE Press (2014)
77. Plankensteiner, K., Prodan, R.: Meeting soft deadlines in scientific workflows using resubmission impact. IEEE Trans. Parallel Distrib. Syst. **23**(5), 890–901 (2012)
78. Preda, N., Kasneci, G., Suchanek, F.M., Neumann, T., Yuan, W., Weikum, G.: Active knowledge: dynamically enriching RDF knowledge bases by web services. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, IN, June 6–10, 2010, pp. 399–410 (2010)
79. Quiroz, A., Huang, E., Ceriani, L.: A robust and extensible tool for data integration using data type models. In: Proceedings of the Twenty-Ninth AAAI, pp. 3993–3998 (2015)
80. Rahman, M., Hassan, M.R., Ranjan, R., Buyya, R.: Adaptive workflow scheduling for dynamic grid and cloud computing environment. Concurr. Comput. Pract. Exp. **25**(13), 1816–1842 (2013)

81. Rheinländer, A., Heise, A., Hueske, F., Leser, U., Naumann, F.: SOFA: an extensible logical optimizer for udf-heavy data flows. Inf. Syst. **52**, 96–125 (2015)
82. Schikuta, E., Wanek, H., Ul Haq, I.: Grid workflow optimization regarding dynamically changing resources and conditions. Concurr. Comput. Pract. Exp. **20**, 1837–1849 (2008)
83. Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., Price, T.G.: Access path selection in a relational database management system. In: Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, pp. 23–34 (1979)
84. Shi, J., Zou, J., Lu, J., Cao, Z., Li, S., Wang, C.: MRTuner: a toolkit to enable holistic optimization for mapreduce jobs. Proc. VLDB Endow. **7**(13), 1319–1330 (2014)
85. Shivam, P., Babu, S., Chase, J.S.: Active and accelerated learning of cost models for optimizing scientific applications. In: VLDB, pp. 535–546 (2006)
86. Simitsis, A., Vassiliadis, P., Dayal, U., Karagiannis, A., Tziovara, V.: Benchmarking ETL workflows. In: TPCTC 2009, 199–220 (2009)
87. Simitsis, A., Vassiliadis, P., Sellis, T.K.: State-space optimization of ETL workflows. IEEE Trans. Knowl. Data Eng. **17**(10), 1404–1419 (2005)
88. Simitsis, A., Wilkinson, K.: Revisiting ETL benchmarking: the case for hybrid flows. In: TPCTC, pp. 75–91 (2012)
89. Simitsis, A., Wilkinson, K., Castellanos, M., Dayal, U.: QoX-driven ETL design: reducing the cost of ETL consulting engagements. In: Proceedings of the SIGMOD, pp. 953–960 (2009)
90. Simitsis, A., Wilkinson, K., Castellanos, M., Dayal, U.: Optimizing analytic data flows for multiple execution engines. In: SIGMOD Conference, pp. 829–840 (2012)
91. Simitsis, A., Wilkinson, K., Dayal, U.: Hybrid analytic flows—the case for optimization. Fund. Inf. **128**(3), 303–335 (2013)
92. Simitsis, A., Wilkinson, K., Dayal, U., Castellanos, M.: Optimizing ETL workflows for fault-tolerance. In: ICDE, pp. 385–396 (2010)
93. Simitsis, A., Wilkinson, K., Dayal, U., Hsu, M.: HFMS: managing the lifecycle and complexity of hybrid analytic data flows. In: ICDE, pp. 1174–1185 (2013)
94. Srivastava, U., Munagala, K., Widom, J., Motwani, R.: Query optimization over web services. In: Proceedings of VLDB, pp. 355–366 (2006)
95. Tan, W., Sun, Y., Lu, G., Tang, A., Cui, L.: Trust services-oriented multi-objects workflow scheduling model for cloud computing. In: ICPCA/SWS, pp. 617–630 (2012)
96. Tao, F., Zhang, L., Laili, Y.: Configurable Intelligent Optimization Algorithm: Design and Practice in Manufacturing. Springer, New York, Incorporated (2014)
97. Tsamoura, E., Gounaris, A., Manolopoulos, Y.: Brief announcement: on the quest of optimal service ordering in decentralized queries. In: Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, July 25–28, 2010, pp. 277–278 (2010)
98. Tsamoura, E., Gounaris, A., Manolopoulos, Y.: Decentralized execution of linear workflows over web services. Future Gener. Comput. Syst. **27**(3), 341–347(2011)
99. Tsamoura, E., Gounaris, A., Manolopoulos, Y.: Optimal service ordering in decentralized queries over web services. IJKBO **1**(2), 1–16 (2011)
100. Tsamoura, E., Gounaris, A., Manolopoulos, Y.: Queries over web services. In: New Directions in Web Data Management, vol. 1, pp. 139–169 (2011)
101. Tziovara, V., Vassiliadis, P., Simitsis, A.: Deciding the physical implementation of ETL workflows. In: Proceedings of the ACM 10th International Workshop on Data Warehousing and OLAP DOLAP, pp. 49–56 (2007)
102. Varol, Y.L., Rotem, D.: An algorithm to generate all topological sorting arrangements. Comput. J. **24**(1), 83–84 (1981)
103. Vassiliadis, P.: A survey of extract–transform–load technology. IJDWM **5**(3), 1–27 (2009)
104. Vassiliadis, P., Simitsis, A., Baikousi, E.: A taxonomy of ETL activities. In: DOLAP 2009, ACM 12th International Workshop on Data Warehousing and OLAP, Hong Kong, November 6, 2009, Proceedings, pp. 25–32 (2009)
105. vom Brocke, J., Sonnenberg, C.: Business process management and business process analysis. In: Information Systems and Information Technology. Computing Handbook, 3rd edn., pp. 26: 1–31 (2014)
106. Vrhovnik, M., Schwarz, H., Radeschütz, S., Mitschang, B.: An overview of SQL support in workflow products. In: Proceedings of ICDE, pp. 1287–1296 (2008)
107. Vrhovnik, M., Schwarz, H., Suhre, O., Mitschang, B., Markl, V., Maier, A., Kraft, T.: An approach to optimize data processing in business processes. In: VLDB, pp. 615–626 (2007)
108. Vu, L.H., Hauswirth, M., Aberer, K.: Qos-based service selection and ranking with trust and reputation management. In: Proceedings of the Cooperative Information System Conference (CoopIS05, pp. 466–483 (2005)
109. Whrer, A., Brezany, P., Janciak, I., Mehofer, E.: Modeling and optimizing large-scale data flows. Future Gener. Comput. Syst. **31**, 12–27 (2014)
110. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, EASE'14, pp. 38:1–38:10 (2014)
111. Yerneni, R., Li, C., Ullman, J.D., Garcia-Molina, H.: Optimizing large join queries in mediation systems. In: ICDT, pp. 348–364 (1999)
112. Zeng, L., Veeravalli, B., Zomaya, A.Y.: An integrated task computation and data management scheduling strategy for workflow applications in cloud environments. J. Netw. Comput. Appl. **50**, 39–48 (2015)
113. Zhou, A.C., He, B., Liu, C.: Monetary cost optimizations for hosting workflow-as-a-service in IaaS clouds. IEEE Trans. Cloud Comput. **4**(1), 34–48 (2016)
114. Zinn, D., Bowers, S., McPhillips, T., Ludäscher, B.: Scientific workflow design with data assembly lines. In: Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science, pp. 14:1–14:10 (2009)