The VLDB Journal (2017) 26:777–801
DOI 10.1007/s00778-017-0477-2

REGULAR PAPER

From conceptual design to performance optimization of ETL workflows: current state of research and open problems

Syed Muhammad Fawad Ali[1] · **Robert Wrembel**[1]

Received: 11 December 2016 / Revised: 14 June 2017 / Accepted: 9 August 2017 / Published online: 6 September 2017

Abstract In this paper, we discuss the state of the art and current trends in designing and optimizing ETL workflows. We explain the existing techniques for: (1) constructing a conceptual and a logical model of an ETL workflow, (2) its corresponding physical implementation, and (3) its optimization, illustrated by examples. The discussed techniques are analyzed w.r.t. their advantages, disadvantages, and challenges in the context of metrics such as autonomous behavior, support for quality metrics, and support for ETL activities as user-defined functions. We draw conclusions on still open research and technological issues in the field of ETL. Finally, we propose a theoretical ETL framework for ETL optimization.

Keywords ETL workflow · ETL conceptual design · ETL logical design · ETL physical implementation · ETL optimization

1 Introduction

A data warehouse (DW) integrates multiple heterogeneous and distributed data sources (DSs) in order to provide a centralized and unified access to data with the end goal of decision support [1]. Data originating from DSs may have different formats and data models, which may not comply with the format and data model of a target DW. Furthermore, incoming data may be inconsistent and of poor quality—ranging from simple spelling errors, missing or inconsistent values, to conflicting or redundant data. Therefore, a special purpose software is typically used in a data warehouse architecture to integrate DSs. This software (a.k.a. process), called Extraction–Transformation–Loading (ETL) is located between data sources and a DW. The first task of an ETL process is to extract data from multiple data sources, typically into a Data Staging Area (DSA). Once data are available in a DSA, the second phase is to perform data quality checks and transformations in order to make data clean and consistent with the structure of a target DW. Finally, the third phase is to load data into a DW.

An ETL process is typically implemented as a workflow, where various tasks (a.k.a. activities or operations), which process data, are connected by data flows [2,3]. The tasks executed in an ETL workflow include among others: (1) extracting and filtering data from data sources, (2) transforming data into a common data model, (3) cleaning data in order to remove errors and null values, (4) standardizing values, (5) integrating cleaned data into one common consistent data set, (6) removing duplicates, (7) sorting and computing summaries, and (8) loading data into a DW. These tasks can be implemented by means of SQL commands, predefined components, or user-defined functions (UDFs) written in multiple programming languages.

There are several proprietary, cf., [4], and open-source, cf., [5], ETL tools available in a business sector for designing and developing ETL workflows. The tools provide proper documentation and graphical user interfaces to design, visualize, implement, deploy, and monitor execution of an entire ETL workflow. However, these tools have a very limited support for designing and developing efficient workflows, since automatic optimization and fine-tuning of an ETL workflow

✉ Syed Muhammad Fawad Ali
fawadali.ali@gmail.com

Robert Wrembel
robert.wrembel@cs.put.poznan.pl

[1] Poznan University of Technology, Poznan, Poland

is not available. Hence, the ETL developer him/herself is responsible for producing an efficient workflow.

This is one of a few reasons that make numerous organizations incline toward in-house development of such ETL tools that best suit their business needs [6,7]. Furthermore, the design of an ETL workflow may become complex, as it consists of multiple activities and each of the ETL activity has its execution cost, which increases with the increase of the volume of data being processed. As a result of a varying cost and a complex design, an ETL workflow may fail amid execution or may not be able to finish its execution within a specified time window. In consequence, a DW becomes outdated and cannot be utilized by its stakeholders. In order to increase the productivity, quality, and performance of ETL workflows some ETL design methods and optimization methods have been proposed.

The ETL research community has proposed several methods for designing a conceptual model of an ETL workflow, which led to its semantically equivalent logical model, physical implementation, and its optimized run-time version. The set of guidelines formulated for the design of a conceptual and a logical model of an ETL workflow [7,8] prompts the automation of a design process, in order to facilitate the development life cycle of the whole DW architecture. Furthermore, community has been focusing on techniques for optimizing the execution of an ETL workflow [9]. The most common techniques are based on tasks rearranging and moving more selective tasks toward the beginning of a workflow, e.g., [10–12]. On top of that, the existing research proves that applying processing parallelism at a data level or at activity level, or both, is a known approach to attain better execution of an ETL workflow.

Since there exist multiple methods and techniques for conceptual, logical, and physical design of an ETL workflow, there is a need of developing a uniform ETL framework, which would: (1) facilitate the ETL developer designing an efficient ETL workflow, by providing hints for optimizing the workflow, and (2) allow the ETL developer to validate and benchmark some alternative workflow designs for given quality objectives.

In this paper, we discuss the state of the art and current trends in designing an ETL workflow and its optimization. The goal of this paper is threefold. First, to study and understand the existing techniques to construct a conceptual and a logical model, its corresponding physical implementation, and optimization of an ETL workflow as well as to evaluate them on the basis of some metrics that we proposed (cf. Sects. 3.5, 6.9). Second, to identify open research and technological issues in the field of designing, implementing, and optimizing an ETL workflow. Third, based on the identified virtues and limitations, to propose a framework for ETL optimization.

This paper is divided into seven sections, each of which starts with an introduction and concludes with the summary of open research and technological issues. Section 2 describes a running example, which we use throughout this paper. Section 3 discusses the findings on a conceptual modeling of an ETL workflow. Section 4 overviews works carried out on designing a logical model of an ETL workflow and approaches to convert a conceptual model into its corresponding logical design. Section 5 introduces techniques for the physical implementation of an ETL workflow. Section 6 focuses on techniques for optimizing an ETL workflow. Finally, Sect. 7 concludes this survey with a summary of open research and technological issues and outlines our ETL design and optimization framework.

2 Running example

To begin our discussion on the existing literature, we will be using the example described in [7], which involves two data sources S1.PARTSUPP (PKEY, SUPPKEY, QTY, COST) and S2.PARTSUPP (PKEY, SUPPKEY, DATE, DEPT, QTY, COST) and a central DW.PARTSUPP (PKEY, SUPPKEY, DATE, QTY, COST). PKEY is the part number, SUPPKEY is the supplier of the part, and QTY and COSTas are the available quantity and cost of parts per supplier, respectively.

Data are propagated from S1.PARTSUPP and S2.PARTSUPP into a DW table DW.PARTSUPP, as shown in Fig. 1.

The example assumes that source S1 stores everyday data about supplies in the European format and source S2 stores the month-to-month data about supplies in the American format. The DW stores monthly data on the available quantity of parts per supplier in the European format, which means that data coming from S2 need to be converted into the European format and data coming from S1 need to be rolled-up at the month level in order to be accepted by the DW. S1 joins data from two separate sources PS1 and PS2, and later the data are transformed into a format accepted by the DW. Data from sources S1 and S2 undergo several transformations, denoted as $A_{11}, \ldots, A_{1n}$ and $A_{21}, \ldots, A_{2n}$, respectively. Finally, data from S1 and S2 are merged at activity A_{nn} to be finally loaded into the DW.

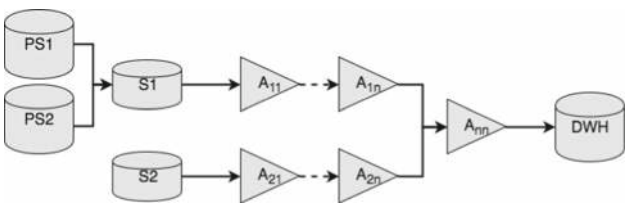

Fig. 1 An ETL workflow for the running example

3 Conceptual model

A conceptual model of a DW serves a purpose of representing business requirements and clearly identifies all business entities participating in a DW. A conceptual model and its documentation help in understanding and identifying data schema and facilitating the ETL developer in transformation and maintenance phase of an ETL workflow.

Until 2002, design, development, and deployment of an ETL workflow were done in an ad hoc manner due to the nonexistence of specific design and development guidelines and standards. The ETL research community has put a lot of effort in formulating the required guidelines, methods, and standards.

This section highlights these methods existing in the literature for a conceptual model of an ETL workflow including graph-, UML-, ontology-, and BPMN-based conceptual models.

3.1 Graph-based conceptual model

A graph-based customizable and extensible conceptual model [7] is among the first approaches in providing formal foundations for the conceptual design of an ETL workflow. The proposed model focuses on the internal structure of the elements involved, interrelationships among sources, target attributes of the elements, and transformations required during loading a DW. The idea behind the proposed framework is to provide the ETL developer with different kinds of transformations required for different ETL scenarios, which cannot be anticipated. Therefore, instead of providing a limited set of transformations, an extensible framework is developed so that the ETL developer can define transformations as required. The paper presents a three-layer architecture for a conceptual model of an ETL workflow that consists of schema layer (SL), meta-model Layer (ML), and template Layer (TL).

SL contains a specific ETL scenario, and all the elements in this layer are instances of ML. ML is a set of generic entities that are able to represent any ETL scenario. Finally, TL enables the generic behavior of the framework, by providing the ETL developer with customizable ETL templates, which he/she can enrich according to different business requirements.

In Fig. 2, SL depicts an ETL workflow of the running example described in Sect. 2. Sources S1.PARTSUPP and S2.PARTSUPP are the instances of class ‘Concept’ in ML and belong to template ‘ER Entity’ (i.e., an entity in an ER model) defined in TL. ‘ER Entity’ is a subclass of ‘Concept’ (from ML). Target DW.PARTSUPP belongs to subclass ‘Fact Table’ in TL. Since a DW stores quantity and cost of parts per supplier in the European format on the daily basis and supplier S2 stores data in the American format,

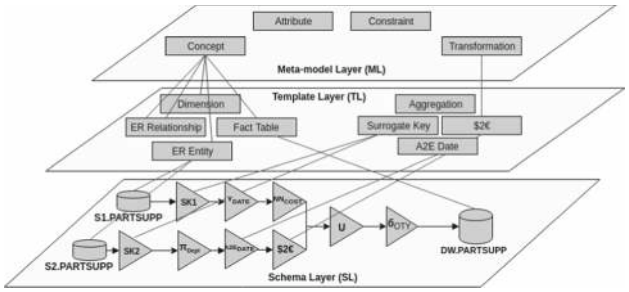

Fig. 2 The multilayer architecture

the customized transformation ‘$2€’ is selected from TL as a subclass of ‘Transformation’ (from ML). The work in [7] describes only the formal foundations for a conceptual model of an ETL workflow. However, much needed design methods and standards were not addressed until [8] proposes a set of steps in order to construct a conceptual model in a managed, customizable, and extensible manner. The set of steps is as follows:

1. Step 1: *identify participating data stores and relationships between them*. In the running example, source data PARTSUPP comes from the union of two concepts S1.PARTSUPP and S2.PARTSUPP. A data store, DW.PARTSUPP is a target concept and stores cost of parts per supplier on a daily basis.
2. Step 2: *identify candidates and active candidates for the involved data stores*. The idea is to include only active candidates in order to keep a conceptual model simple. For example, let us assume in the running example, S2.PARTSUPP can be populated either by file F1 or F2. So, F1 and F2 are candidate data stores for concept S2.PARTSUPP. The ETL developer chooses F2 due to business requirements to populate S2.PARTSUPP, and thus, F2 becomes the active candidate. Once the active candidates are identified, the ETL developer can choose not to show F1 in a conceptual model. However, it does not mean F1 is eliminated from a conceptual model; it remains a part of the model to be used in the later stages.
3. Step 3: *identify the need of data transformation*. If a transformation is required, define mapping rules between source and target concepts. A transformation can be a surrogate key assignment, conversion of units, or just a simple mapping of attributes. In the running example, S2.PARTSUPP has values in the American format, whereas DW.PARTSUPP accepts the European format values. Therefore, attributes COST and DATE from S2.PARTSUPP have to be transformed into Euros and the European date format, respectively, to populate DW.PARTSUPP.
4. Step 4: *annotate the model with run-time constraints*. Annotation can be done using notes, which correspond

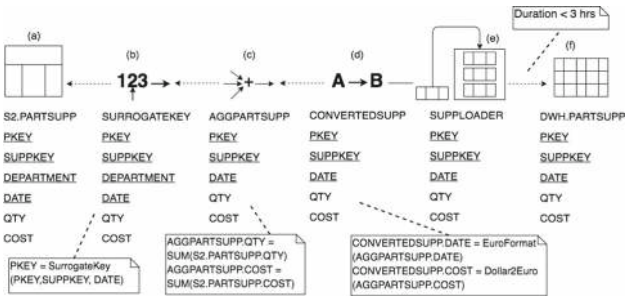

Fig. 3 A UML-based design of an ETL workflow

to a particular operation, concept, or a relationship in a conceptual model. For example, DW.PARTSUPP needs to be populated within a particular time window, e.g., three hours. Therefore, to state this constraint, the ETL developer attaches a note at the particular operation to specify the run-time constraint.

3.2 UML-based conceptual model

The unified modeling language (UML) [13] is a standard modeling language in the field of software engineering in order to visualize the design of systems in a standardized way. [14] point out that methods [7,8], discussed in Sect. 3.1, may result in a complex ETL workflow design due to the absence of a standard modeling language and treating attributes as 'first-class citizens' in the model. Therefore, UML is used as a standard modeling language for defining the most common ETL activities (including among others: data integration, transformation of attributes between source and target data stores, as well as generation of surrogate keys). The ETL activities are represented by UML packages to model a large ETL workflow as multiple packages, thus simplifying the complexity of an ETL workflow for the ETL developer.

Figure 3 represents the lower flow of the running example as a UML-based conceptual model of an ETL workflow using the defined stereotype icons. For example, (a) is a 'Table' stereotype icon, which is used to represent data source S2.PARTSUPP, (b) represents surrogate key assignment, (c) represents aggregation of data in S2.PARTSUPP, (d) represents conversion of attribute DATE from the American to the European format and conversion of attribute COST from Dollars to Euros, (e) represents loading data, and (f) represents DW.PARTSUPP. The design and run-time constraints are also shown using UML 'Note' artifact.

3.3 Ontology-based conceptual model

The approaches discussed in Sects. 3.1 and 3.2 require the ETL developer to manually derive the ETL transformations and inter-attribute mappings at a conceptual level of an ETL workflow. The work in [15] proposes a semiautomatic method for designing a conceptual model of an ETL workflow, leveraging an ontology-based approach. The proposed approach uses ontology instead of UML because ontology is capable of deriving ETL transformations automatically using ontology 'reasoners.' The proposed solution facilitates the construction of an ETL workflow at a conceptual level and deals with the problem of semantic and structural heterogeneity.

As the first step toward creating an ontology, a common vocabulary is constructed. To this end, the ETL developer has to provide the information about the application domain and user requirements about a DW, for example, primitive concepts and their attributes, possible values of the attributes, and relationship among the concepts and attributes. Using the information provided by the ETL developer, the vocabulary is generated, which consists of the following elements: (1) V_C—concepts involved in the workflow, (2) V_P—a set of attributes that characterizes each concept, (3) V_F—various types of formats that may be used for the attributes, (4) V_T—a set of allowed values that an attribute may take, (5) f_P—a function associating each attribute in V_P to primitive concept V_C it describes, (6) f_F—a function associating each representation format V_F to attribute in V_P, and (7) f_T—a function associating each value to representation format V_F, or directly to attribute in V_P.

In the running example, V_C is *PARTSUPP*, V_P is a set of attributes {pPKEY, pSUPPKEY, pDATE, pQTY, pCOST}, V_F for pCOST is $V_F cost$ ={Dollars, Euros}, V_F for pDATE is $V_F date$ ={American, European}, and V_F for pPKEY is $V_F pkey = \{source_p key, dwh_p key\}$. Let us assume, a DW also stores data of parts type pTYPE 'small,' 'medium,' and 'large,' then V_T for pType will be $V_T Type = \{small, medium, large\}$.

After the common vocabulary is formulated, the second step is to annotate the data sources based on the constructed vocabulary. The annotation for source S2 is given in Table 1, which provides the following eight items: (1) ϕ—the format of an attribute, (2) *min*—the minimum value for an attribute, (3) *max*—the maximum value for an attribute, (4) T—the set of values an attribute has on a given relation, (5) n—the cardinality of an attribute in the relation, (6) R'—the foreign key of a relation, (7) X_f—the aggregation function for example 'sum,' max,' or 'avg,' and (8) X_p—the property on which the aggregation is based.

Once the application vocabulary and the annotations are described, the third step is to construct an application ontology. It describes the application domain, relationships, as well as mappings between sources and a target. The application ontology consists of: (1) a set of primitive classes similar to the specified concepts, (2) representation formats, and (3) ranges or sets of values as defined in the vocabulary. Finally, the constructed ontology is used to generate a

Table 1 Annotation for data store S2

S2	ϕ	min	max	T	n	R'	$\mathbf{X}_f$	$\mathbf{X}_p$
$I_p PKEY$	$source_p key$	–	–	–	1	–	–	–
$I_p SUPPKEY$	–	–	–	–	1	–	–	–
$I_p DATE$	*American*	–	–	–	1	–	–	–
$I_p QTY$	–	–	–	–	1	–	*Sum*	*SUPPKEY*
$I_p COST$	*Dollars*	–	–	–	1	–	*Sum*	*SUPPKEY*
$I_p TYPE$	–	–	–	{*small*, *medium*, *large*}	1	–	*Sum*	*SUPPKEY*

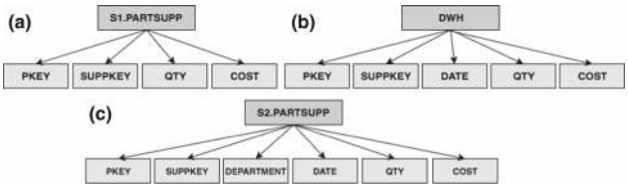

Fig. 4 A data-store graph for data sources and a DW

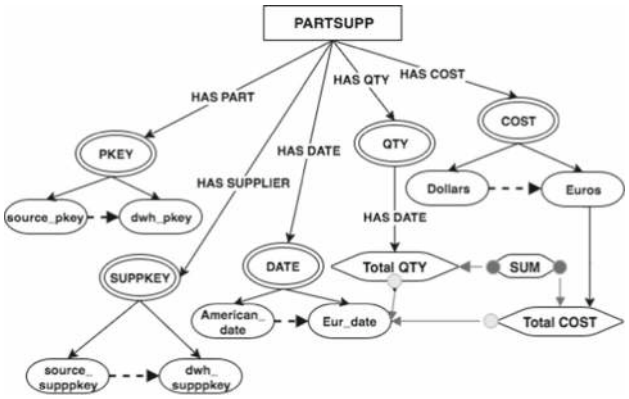

Fig. 5 The ontology graph

conceptual model automatically using the OWL 'reasoner.' This solution enables the ETL developer to explicitly and formally represent a conceptual model using Ontology Web Language-Description Logic (OWL-DL).

OWL [16] helps in creating a flexible model that can be redefined and reused during different stages of a DW design. The well-defined semantics allows automated reasoning. This solution is applicable to the relational databases only; however, a DW may also contain semi-structured and unstructured data.

To provide the support for semi-structured data, [17] propose a solution, which is an extension of the work presented in [15] to cater both structured and semi-structured data sources. The proposed approach uses graphs to represent a conceptual model for data sources in order to handle both structured and semi-structured data in a uniform way. The schema presented as a graph is called a data-store graph (DSG). For a relational schema, a DSG is designed as follows: (1) Nodes represent elements of a schema, (2) edges represent relationships among the elements, (3) labels on each edge represent min and max cardinalities of a reference, and (4) leaf nodes represent elements containing data.

To construct a DSG from semi-structured data in order to generate a conceptual model using an ontology, let us consider addition of a complex-type attribute DEPARTMENT derived from LOCATION (CITY, BRANCH) into source S2 of the running example. Figure 4 shows the DSGs constructed by following the proposed steps to convert an XML document into a DSG, which are as follows: (1) Nodes represent elements, attributes, as well as complex and simplex types in an XML document, (2) edges represent nesting or referencing of XML elements, and (3) labels represent min and max cardinalities of an XML element. Figure 4b illustrates a DSG that depicts the DEPARTMENT as a complex type.

Figure 5 illustrates the ontology graph, where PARTSUPP has attributes PKEY, SUPPKEY, DATE, QTY, and COST. For each of these attributes, a corresponding attribute and a class are created in the ontology. Since PKEY has to be transformed into a surrogate key, a separate class along with its property is created in the ontology graph. In case of COST, separate classes 'Dollars' and 'Euros' are introduced. Similarly, for the case of date format, classes 'AmericanDate' and 'EuropeanDate' are created in the ontology graph. Classes 'TotalCost' and 'TotalQty' are also introduced to represent the aggregated costs and quantity, according to the assumptions taken in the running example.

The ontology graph and a DSG are then used to annotate each data store by defining mappings between these two graphs. Finally, the semantic annotations are used along with an application ontology to infer a set of generic transformations to construct a conceptual model of an ETL workflow.

In the preceding approaches [15,17] related to an ontology-based conceptual model, the ETL developer is responsible for manually sketching the required mappings and transformations of schema from a source to a target data store.

To reduce the manual work required by the ETL developer, [18] propose a semiautomatic approach to build an ETL workflow in a step-by-step manner through a series of customizable and extensible set of graph transformations rules. These rules are based on the already provided ontology in

order to determine which ETL operators are applicable in the initial graph, i.e., a graph generated after converting a semi-structured data. The final graph, i.e., a graph generated after applying transformation rules, depicts a conceptual model of an ETL workflow with an appropriate choice of ETL activities.

3.4 BPMN-based conceptual model

Besides the aforementioned approaches proposing conceptual models (cf. Sects. 3.1, 3.2, 3.3), the work presented in [19] proposes a Business Process Model Notation (BPMN) to create a platform-independent conceptual model of an ETL workflow.

The paper discusses several BPMN operators to represent various ETL activities. For example, BPMN gateways represent the sequence of activities in an ETL workflow, based on conditions and constraints; BPMN events represent start, end, and error handling events; BPMN connection objects represent the flow of activities; BPMN artifacts describe the semantics of an ETL task.

Figure 6 illustrates the running example, which has three swim lanes, namely 'Extract,' 'Transform,' and 'Load.' S1.PARTSUPP is populated using tables PS1 and PS2. Data source S2.PARTSUPP requires data from text file F2.txt. First, data are extracted from tables PS1, PS2, and file F2.txt, depicted as operations 'Extract from PS1,' 'Extract from PS2,' and 'Extract from F2.txt,' respectively. Then, extracted data are loaded into temporary tables using operations 'Load S2.PARTSUPP' and 'Load S1.PARTSUPP.' These activities can be executed in parallel since there is no dependency between them. Once the data are extracted and loaded into staging tables, a transformation is performed according to the business requirements.

The transformation part is depicted as a BPMN collapsed sub-process. Sources S1.PARTSUPP and S2.PARTSUPP in swim lane 'Transform' also have a check to reject all

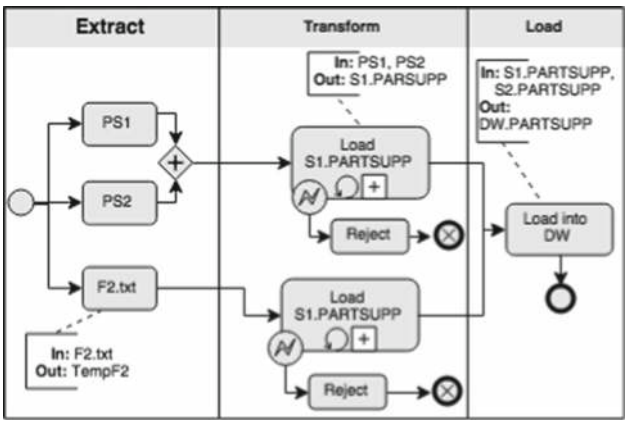

Fig. 6 The BPMN representation of an ETL workflow

the rows whose COST is NULL, which is handled by a BPMN error handling event. Finally, the data are loaded into DW.PARTSUPP using operation 'Load into DW,' as shown in the 'Load' swim lane.

After the BPMN design is completed, the model is translated into a Business Process Execution Language (BPEL), which we will discuss in Sect. 5. This work enables the design developed in BPMN to be compatible across multiple tools and easy to extend to fit the requirements of a particular application. The BPMN approach was then adapted by [20–22] to construct a conceptual model of an ETL workflow.

The work in [22] proposes a layered method that starts with business requirements and systematically converts a conceptual model into its semantically equivalent physical implementation. The entire method is based on the QoX—suit of quality metrics [23] to construct an optimal ETL workflow. The QoX metrics are considered during the design and development of ETL workflows ranging from quantitative to qualitative metrics (e.g., performance, recoverability, and freshness). [22] fill the gap between different stages (conceptual, logical, and physical) of an ETL workflow design. Once the conceptual design is expressed in BPMN, it is converted into XML to translate a conceptual design into its semantically equivalent logical model. The logical model is then used to optimize an ETL workflow design and for creating the corresponding physical model.

The approach presented in [20] uses BPMN and model driven development to specify an entire ETL workflow in a vendor-independent way and to automatically generate the corresponding code in different commercial tools. Once an ETL workflow code is generated, a 4GL grammar is used in order to generate a vendor-specific code.

The work in [21] complements and extends the work of [20,22] by incorporating specific conceptual model constructs as BPMN patterns for ETL activities like 'change data capture,' 'slowly changing dimensions,' 'surrogate key pipelining,' or 'data quality coverage.' This foundation can be extended to build more BPMN patterns covering all the activities of an ETL workflow, which results in helping develop high-quality, error-free, and an efficient ETL workflow.

3.5 Summary

In Sect. 3, we have discussed different techniques and methods for representing a conceptual model of an ETL workflow, which include graphs, UML, ontology, and BPMN. Below, we summarize the approaches on the basis of the following criteria:

– *Autonomous behavior*—whether a design is manual, automatic, or semiautomatic (i.e., how much input it requires from the ETL developer);

– *DSs format*—what kind of data sources are supported, i.e., structured, unstructured, or semi-structured;
– *UDF support*—whether user-defined functions are supported;
– *Quality metrics*—whether quality metrics guide the design of an ETL workflow;
– *Unified model*—whether an ETL design is easily translated from a conceptual model into its semantically equivalent logical model and whether it can be implemented using any ETL framework.

1. **Graph-based approaches** [7,8]

Pros:
- Widely accepted graph-based models are used, which help the ETL developer to outline a conceptual model of an ETL workflow in a standardized way.
- They present the first steps toward translating a conceptual model of an ETL workflow into its semantically equivalent logical model.

Cons:
- *No autonomous behavior*—the ETL developer has to manually derive ETL transformations and inter-attribute mappings at a conceptual level.
- *Structured data only*—only the structured input data sources are supported, and there is no discussion on how to handle unstructured or semi-structured data sources.
- *No UDF support and quality metrics*—the operators and templates for traditional ETL tasks are proposed; there is no support for UDFs; quality metrics are not taken into consideration while constructing a conceptual model.
- *Challenges*—an ETL design may become complex due to the absence of a standard modeling language and treating attributes as 'first-class citizens' in the model.

2. **UML-based approach** [14]

Pros:
- *Unified model*—a method is proposed to standardize the conceptual design of an ETL workflow (UML is a standard modeling language).

Cons:
- *No autonomous behavior*—the ETL developer has to provide input at each step of the conceptual design.
- *Structured data only*—only the structured input data sources are supported.
- *No UDF support and quality metrics*—no support for user-defined functions and quality metrics.

3. **Ontology-based approaches** [15,17,18,24]

Pros:
- *Semi-autonomous behavior*—the ontology-based models propose semiautomatic methods to design a conceptual model of an ETL workflow in a step-by-step manner (based on reasoners on ontologies, it is possible to derive ETL transformations automatically).
- *Structured & semi-structured data*—[15,18] focus on structured data only and [17] focus also on semi-structured data.

Cons:
- *No UDF support and quality metrics*—UDFs are not supported; quality metrics are not taken into consideration while constructing a conceptual ETL model.
- *Unified model*—an ontology-based conceptual design cannot be directly translated into its semantically equivalent logical model. It requires a fair amount of effort from the ETL developer to translate the design.
- *Challenges*—manually creating an ontology and defining the relationships among the ontology elements is a difficult and time-consuming task. Constructing an ontology manually requires high correctness and detailed description of data sources, thus if an ontology is created manually it becomes more prone to errors.

4. **BPMN-based approaches** [19–22]

Pros:
- *Semi-autonomous behavior*—[21] introduce various BPMN patterns as constructs for frequently used ETL operators and activities.
- *Quality metrics & unified model*—[22] propose a systematic method to translate business requirements into a conceptual model and conceptual model into its semantically equivalent logical model, based on quality metrics. [20] use BPMN and model driven development approach to develop vendor-independent design of an ETL workflow.

Cons:
- *No UDF support*.
- *Challenges*—converting conceptual model into its equivalent logical and physical implementation requires ETL developers to have specific

knowledge and hands-on experience in BPMN and BPEL.

To conclude, the graph-based models can be used to represent a complex conceptual design of an ETL workflow by using standard notations, whereas, for simpler ETL workflows, approaches based on UML, ontology, and BPMN are well suited. Such models reflect business requirements as well as provide technical perspective of the problem. Nonetheless, there is a need for a single agreed unified model, easy to validate and benchmark an ETL design for its quality objectives. Also all the discussed approaches require the ETL developer to extensively provide some input during the design phase of an ETL workflow, as well as require technical knowledge from business users to understand and validate an ETL design. Furthermore, despite the fact that multiple approaches have been proposed, the research community has not yet agreed upon the standard notation for representing a conceptual model of an ETL workflow.

4 Logical model

The next step in ETL development life cycle is a logical design of an ETL workflow. A logical design describes detailed description of an ETL workflow such as relationships among the involved processes with participating data sources, a description of primary data flow from source data stores into a DW, including an execution order of ETL activities as well as an execution schedule of an entire ETL workflow. A recovery plan and a sequence of steps in case of recovery from a failure are also devised during a logical design of an ETL workflow.

In this section, we will discuss approaches to a logical design of an ETL workflow.

4.1 Graph-based logical model

Initial approaches to designing a logical model of an ETL workflow are based on graphs. The work presented in [25] proposes a formal logical model as a graph called *Architecture Graph*. The graph shown in Fig. 7 illustrates an ETL workflow as a set of ETL activities and a flow of data between these activities. The nodes in the graph represent ETL activities, record sets, and attributes, whereas the edges represent different types of relationships among ETL activities. For example, Fig. 7 illustrates activities 'SK_T' and '$2€.' Parameter 'PKEY' is mapped to attribute 'PKEY' of activity 'SK_T' and parameter 'SKEY' is mapped to attribute 'SKEY' via regulator relationship. The regulator relationship denotes that external data provider is used to populate the attribute. The provider relationships denote the data flow from source record sets toward the target record set. The part-of relationship denotes the relationship between

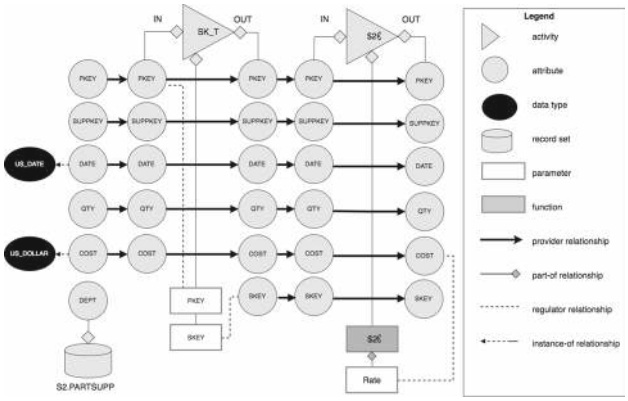

Fig. 7 Architecture Graph of an ETL workflow

attributes and activity, record set, or function. The instance-of relationship describes a relationship between data types and attributes.

[25] propose the following steps to construct a logical model of an ETL workflow as the Architecture Graph:

1. Incorporate structured entities, i.e., activities and record sets in a graph along with all the attributes. For example, Fig. 7 illustrates S2.PARTSUPP as a structured entity along with its attributes, i.e., DEPT, COST, QTY, DATE, SUPPKEY, and PKEY.
2. Connect 'Activity nodes' with their respective attributes through a part-of relationship. For example, in Fig. 7 'Activity nodes' 'SK_T' and '$2€' are connected with their respective attributes as the part-of relationship that is depicted as a connector with a diamond. The 'IN' and 'OUT' labels on an activity represent that the attributes belong to the input and output schema of an activity, respectively.
3. Incorporate data and function types using instance-of relationships. For example, data types 'US_Date' and 'US_Dollar' are connected to their respective attributes 'DATE' and 'COST' as instance-of relationships, depicted in Fig. 7.
4. Construct the regulator relationships. For example, in Fig. 7, regulator relationships among the parameters of the activities and attributes are depicted with simple dotted edges.
5. Establish provider relationships that capture the flow of data from source to target. For example, the data flow from source attributes toward target attributes is depicted as bold solid arrows in Fig. 7.

The proposed graph is considered as a formal logical model of an ETL workflow. In [26,27], the authors used the aforementioned model along with a formal Logical Data Language (LDL) to define the semantics of each ETL activity. [27] also mention the reusability framework to com-

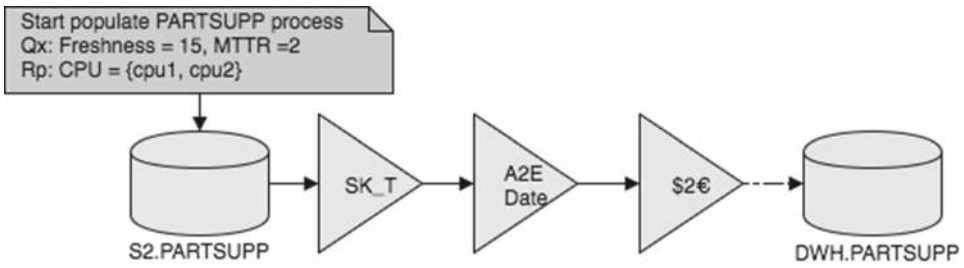

Fig. 8 Parameterized DAG (DAG-P) for the example ETL workflow

pliment the generic behavior of the proposed model, which is achieved through a meta-model layer (ML) and a template layer (TL) as discussed in [7] (cf. Sect. 3.1). The authors also propose a user-friendly graphical tool to facilitate the ETL developer to design an ETL workflow.

[22] propose to construct a logical model of an ETL workflow using a method to parameterize a Directed Acyclic Graph (DAG), called here *Parameterized DAG* (DAG-P). The DAG is created by translating a BPMN-based conceptual model, as mentioned in Sect. 3.4. DAG-P operations, transformations, and data stores are represented as vertices of the graph. Edges represent data flows from a source data store to a target data store. The parameters in DAG-P are used to incorporate business requirements [23], physical resources (needed for an ETL workflow execution), and other generic characteristics (such as visualization) of an ETL workflow. Figure 8 shows a part of the running example as DAG-P. In this figure, data are extracted from S2.PARTSUPP, 'SK_T' generates a surrogate key, 'A2E Date' converts the DATE to European format, '$2€' converts the COST to Euros, and finally, data are loaded into DW.PARTSUPP. The parameters depict that the 'Populate PARTSUPP Process' should run every 15 min, has a mean time to recover MTTR 2 min, and uses 2 CPUs. The DAG-P logical model is represented using XML notation, called the xLM model.

The xLM model describes different naming standards to represent DAG-P in an XML notation. For example, $<design/>$ represents all the elements in an ETL graph, $<node/>$ represents a vertex, and $<edge/>$ represents a data flow in an ETL graph connecting two vertices. The $<properties/>$, $<resources/>$, and $<features/>$ represent different parameters to identify QoX metrics in an ETL workflow.

4.2 From conceptual to logical model

The work presented in [28] proposes a set of steps to transform a conceptual model of an ETL workflow to its corresponding logical model. The models are represented by graphs called *Conceptual Graph* and *Architecture Graph*, respectively. The following steps map a conceptual model into a logical model:

1. Identify data stores and transformations required in an ETL workflow and describe inter-attribute mappings between source and target data stores.
2. Determine 'Stages' to identify the proper order of activities in a conceptual model to assure a proper placement of activities in a logical model.
3. Follow the following five-step method in order to translate a conceptual model into its corresponding logical model:
   (a) Simplify a conceptual model such that only required elements are present in the model.
   (b) Map the concepts of a conceptual model into data sources in a logical model such that part-of relationships do not change. The part-of relationship denotes the relationship between attributes and activity, record set, or function.
   (c) Map transformations defined in a conceptual model to logical activities and then determine the order of execution of the ETL activities.
   (d) Represent ETL constraints with separate activities in a logical model and determine their execution order.
   (e) Generate a schema involved in a logical model using the algorithm proposed in [12] in order to assure that semantics of the involved concepts does not change even after changing the execution order of tasks in an ETL workflow.

As discussed in Sect. 3.4, the work in [22] proposes a method that covers all stages of an ETL workflow design, i.e., from gathering business requirements to designing a conceptual model and finally translating it into an XML-based logical model as a DAG-P. The reason behind choosing XML is its ability to easily transform one XML model (conceptual) into another XML model (logical). The paper proposes a semiautomatic approach to convert a conceptual model represented using XPDL into the xLM model (XML representation of a logical workflow). For example, the XPDL workflow is mapped into xLM $<design/>$, XPDL transitions—into xLM $<edges/>$, XPDL activities—into xLM $<nodes/>$. The XML representation of a logical model is used for creating a physical model of an ETL workflow, which is discussed in Sect. 5.

4.3 Summary

In this section, we have discussed a graph-based logical representation of an ETL workflow and steps to translate a conceptual model into its semantically equivalent logical model. We have outlined step-by-step methods to formulate a graph-based logical model of an ETL workflow. The discussed methods are not trivial to adopt and require a substantial amount of manual effort and background knowledge from the ETL developers. Below, we summarize the approaches on the basis of the criteria described in Sect. 3.5.

1. **Graph-based approaches** [22,25–27,29]

   Pros:
   - *Quality metrics*—[22] propose annotations to incorporate quality metrics in an ETL workflow for its efficient and reliable execution.
   - *Unified model*—[27] propose a reusable framework that supplements a generic behavior of a logical model by defining semantics of each ETL activity in a graph. [22] propose a logical model as a graph, which is implemented in XML and can be used in any XML-based framework.

   Cons:
   - *No autonomous behavior*—the discussed methods require the ETL developer either to manually construct a logical model from a given conceptual model or to provide an extensive amount of input to the system, to generate a logical model from a conceptual model.
   - *No UDF support*.
   - *Challenges*—the discussed approaches demand a substantial amount of input from the ETL developer. For example, such an input is required in: (1) the task of identifying stages (c.f. Sect. 4.2) to make sure the activities are in a proper order and (2) the task of defining the mappings to translate a conceptual model to its equivalent logical model. Such tasks are not trivial in nature and are prone to errors.

From the above discussion, we can conclude that there still exists a need to develop a fully or semiautomatic intelligent system that would guide the ETL developer to produce a logical design of an ETL workflow satisfying some predefined quality criteria.

5 Physical implementation

Having developed a conceptual and a logical model of an ETL workflow, its physical model has to be produced. A physical model describes and implements the specifications and requirements presented in a conceptual and a logical model.

5.1 Implementation based on reusable templates

The work in [30] proposes a method for mapping a logical model of an ETL workflow into its corresponding physical model. A logical model is formulated as a state-space problem, where states are a set of physical level scenarios and each state has a different cost. An initial state of the state-space problem is generated by converting each logical activity to its corresponding physical activity using a library of reusable templates. The library consists of both logical- and physical-level templates. The templates include a set of properties, require some input parameters, and thus are able to be customized according to a particular ETL scenario.

Let L_t denote a set of logical templates and P_t be a set of physical templates in a template library. Consider logical activity \$2€ that converts dollars to euros for S2.PARTSUPP in the running example. First, logical template 'CurrencyConvert_L_t' (to convert currency amount) is picked from L_t. Then, an input and output schema and an attribute over which the conversion will take place are assigned to the selected template. For example, the required input and output schema will be S2.PARTSUPP (PKEY, SUPPKEY, DATE, QTY, COST, SKEY) and attribute COST will be passed for conversion. Finally, the selected template 'CurrencyConvert_L_t' is mapped into a valid physical template from P_t.

Multiple versions of the solution can be generated by selecting different physical templates, provided all constraints and conditions of the selected physical template are satisfied.

5.2 Implementation based on BPEL

A BPMN approach [19] discussed in Sect. 3.4 implements a BPMN-based conceptual model into Business Process Execution Language (BPEL). BPEL is a standard executable language for specifying interactions with Web services, based on XML. BPEL has four main sections: 'partnerLinks,' 'variables,' 'faultHandlers,' and 'process.' In order to translate BPMN into BPEL, first the basic attributes are mapped such as business process name and related namespaces are mapped to the 'process name' in BPEL. Then, ETL tasks in a BPMN are represented as the type of 'services' and are mapped into the 'partnerLinks' in BPEL.

For example, Fig. 9, which is an extract from Fig. 6, illustrates ETL activities as (extract from) 'PS1,' (extract from) 'PS2,' and (extract from) 'F2.txt.' These activities are mapped to 'partnerLinks' in BPEL. BPMN properties that support an ETL workflow such as 'TempF2' in Fig. 6 are

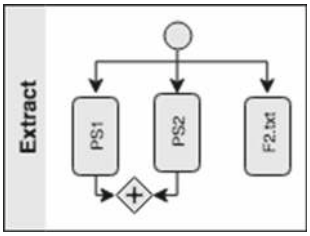

Fig. 9 BPMN representation of an 'Extract' Step

stored in BPEL 'variables.' The error end events in a BPMN are mapped to 'faultHandlers' event in BPEL. Finally, the 'process' section in BPEL contains the description of the ETL activities that are included in an ETL workflow.

5.3 Implementation based on XML

[22] propose to model an ETL workflow as an ETL graph encoded in XML representation (as described in Sects. 3, 4). To translate an XML-based logical model to its corresponding physical implementation, the authors use an appropriate parser. For example, an XML encoded logical model can be translated into a physical implementation format understandable by Pentaho Data Integrator (PDI), as follows:

1. element $<design/>$ maps into 'job' activity, and if $<design/>$ element is nested, then it maps into 'transformation' activity in PDI,
2. element $<node/>$ maps into 'step,'
3. elements $<name/>$ and $<optype/>$ map into 'name' and 'type' of 'step,' respectively,
4. element $<type/>$ of node describes the type of an ETL activity, e.g., a data store or an ETL operator in 'step,'
5. element $<edge/>$ specifies the order and interconnection of 'step,'
6. element $<properties/>$ specifies the physical properties of the 'step.'

'job,' 'step,' 'name,' and type of step' are artifacts in PDI. Hence, using the aforementioned mapping rules and the appropriate parser, the physical implementation of a XML encoded logical model is easily generated.

5.4 Summary

In this section, we have examined the methods and techniques to translate logical models of an ETL workflow into their corresponding physical implementations using reusable templates, engine-specific XML parser, and BPEL. The advantages and disadvantages of the discussed approaches can be summarized based on the metrics described in Sect. 3.5 as follows.

1. **Reusable templates approach** [30]

   Pros:
   - *Semi-autonomous behavior*—the techniques that physically implement a graph-based logical model use a library of reusable (possibly error-free and efficient) templates; a number of an ETL workflow variants may be generated by selecting different physical templates.
   - *UDF support*—UDFs are supported as black-box ETL activities and are considered during implementation and performance optimization.

   Cons:
   - *Structured data*—only structured data sources are supported.
   - *Quality metrics*—only execution and performance cost as quality metrics are considered.
   - *Unified model*—the ETL developer has to manually translate a logical model into its corresponding physical implementation if a template is not already provided for a certain ETL activity. Furthermore, the templates are platform dependent, which limits their application.
   - *Challenges*—a limited set of logical and physical templates is provided.

2. **BPEL-based approach** [19]

   Pros:
   - *Semi-autonomous behavior*—BPEL is used to physically implement a BPMN-based logical model; mapping rules are required to implement a BPMN-based model into BPEL.
   - *Unified model*—the physical implementation is done using BPEL and thus is platform independent as an ETL processes can be exposed as a Web service.

   Cons:
   - *Structured data*—only structured data sources are supported.
   - *No UDF support*.
   - *Challenges*—the ETL developer must have prior knowledge of BPEL and tools that support BPEL-based ETL workflows.

3. **XML-based approach** [22]

   Pros:
   - *Semi-autonomous behavior*—a step-by-step method is proposed to generate a physical implementation of an XML-based logical model using engine-specific XML parser, but the ETL developer has to provide the mapping rules to implement an ETL workflow.

- *Quality metrics*—the performance, freshness, recoverability, and reliability quality metrics are addressed.
- *Unified model*—the logical design is created in the XML format. Since most of the ETL tools support XML, it is easy to generate a corresponding physical implementation using existing tools.

Cons:

- *Structured data*—only structured data sources are supported.
- *No UDF support*.
- *Challenges*—generating a physical implementation of a workflow from its XML-based logical model requires a set of carefully defined rules.

To conclude, the discussed methods require extensive amount of input from the ETL developer to execute a translation of a logical model into its physical representation. Although a few approaches have been proposed in this field, there is a need for a framework that automatically or semi-automatically translates a logical model into its physical implementation with minimum or no human support.

6 Optimization of an ETL workflow

As discussed in Sect. 1, an ETL workflow has a complex structure because it comprises many different ETL activities, which may be implemented in various ways, e.g., relational operators or UDFs. Each of these activities may have fluctuating execution time that increases with respect to the size of incoming data. Therefore, minimizing the execution time is of particular importance. In this section, we will discuss different research approaches and commercial tools that support performance optimization of ETL workflows.

6.1 State-space approach for optimizing an ETL workflow

[12] present a concept to reduce the execution cost either by decreasing the total number of activities or by changing the order of activities in an ETL workflow. To this end, a state-space search problem is defined, where each state in a search space is a Directed Acyclic Graph (DAG). In a DAG, activities are represented as graph nodes and relationships among nodes are represented as directed graph edges. To find an optimal ETL workflow, new states are generated that are semantically equivalent to the original state. A transition from an original state to a new state may involve *swapping* two activities, *factorizing*/*distributing* two activities, *merging*, or *splitting* activities.

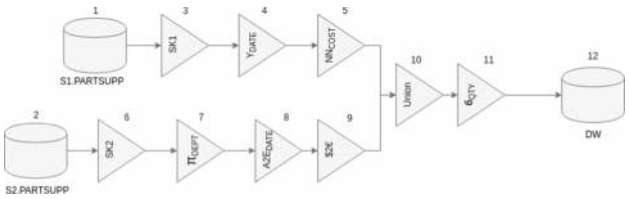

Fig. 10 ETL workflow before applying operation *distribute*

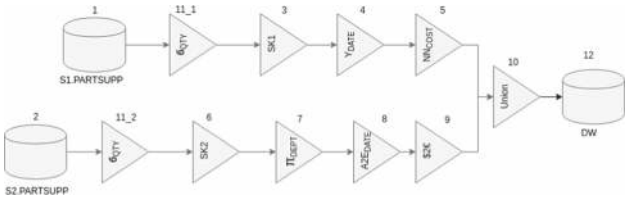

Fig. 11 ETL workflow after applying operation *distribute*

To illustrate the generation of a semantically equivalent new state using operation *distribute* (i.e., to distribute the data flow in an activity into different data flows rather than operating over a single data flow), consider a conceptual model of the running example as depicted in Fig. 10. The data are propagated into a DW in two parallel flows passing through different activities and are finally unified at Activity 10. Then, in Activity 11, the flow checks for the value of attribute QTY before loading data into a DW. This activity is highly selective; therefore, it is beneficial to push the activity to the beginning of the flow and distribute the activity into two parallel flows. Figure 11 shows Activity 11_1 and Activity 11_2 after applying the *distribute* operation to Activity 11 in Fig. 10. This approach reduces the total cost of the flow without changing the semantics of the ETL workflow.

A work of [31] extends that of [12] w.r.t. generating an optimal ETL workflow in terms of performance, fault tolerance, and freshness, as described in [23]. In order to achieve quality objectives, the approach applies three new transitions, namely *partition*, *add_recovery_point*, and *replicate*. *partition* is used to parallelize an ETL workflow to achieve better performance. *add_recovery_point* and *replicate* are used to provide a workflow persistence and recovery in case of a failure.

To generate a search space, the 'exhaustive search (ES)' algorithm is used [12]. Next, the search space is pruned by using a cost model and different heuristics for performance, reliability, and recoverability metrics. Once the state-space search problem is constructed using the ES algorithm, heuristics and greedy algorithms are used to reduce and explore the search space to get an optimal ETL workflow.

[30] also model an ETL workflow as a state-space search problem and apply *sorters* in a graph node. Sorters change the order of input tuples, because in some cases the order plays an important role in an improved execution of an ETL

activity. In order to obtain the optimal solution, the 'exhaustive ordering (EO)' algorithm is used. EO takes a logical design of an ETL workflow as an input in the form of a DAG and computes its *signature* and its computing cost. The signature is a string representation of a physical design of an ETL workflow. To represent a workflow (i.e., a graph) as a string, the following rules are proposed: **a@p**—the physical implementation of 'p' of logical activity 'a,' **.** (dot)—names of activities forming a linear path separated by dot (.), **//**—concurrent activities delimited by '//' and each path enclosed in parenthesis, **a_b(A,B)**—a sorter placed among activity 'a' and 'b' based on attributes 'A' and 'B,' **V!A**—a sorter on table 'V' based on attribute 'A.' Based on the rules, an ETL workflow shown in Fig. 10 can be represented in terms of the following signature:

((1.3.4@DT.5@NN) // (2.6.7@PO.8.9)). 10@NL.11.12

In the signature, activities 1, 3, 4, and 5 (in Fig. 10) form a linear path in the upper level flow and activities 2, 6, 7, 8, and 9 form a linear path in the lower level flow. Both the upper and the lower level flows are concurrent and therefore are separated by '//.' *((1.3.4@DT.5@NN)* denotes that activity 4 is aggregated based on the date function 'DT' and activity 5 performs a not null check 'NN.' *(2.6.7@PO.8.9)).10@NL.11.12* denotes that activity 7 applies projection 'PO' on attribute DEPT as it is not required by a DW. Finally, both flows are merged based on nested loop 'NL' at activity 10.

Once the signature is computed, the EO algorithm generates all possible states by placing sorters at all possible positions. The EO algorithm then uses all possible combinations of different physical implementations for each activity. Finally, it chooses a state with minimum execution cost as the optimal physical implementation.

6.2 Dependency graph for optimizing an ETL workflow

The optimization concept contributed in [11] draws upon the idea of rearranging tasks (activities) in an ETL workflow (as proposed in [12]), in order to construct a more efficient variant of this workflow. The following assumptions are made in [11]:

- an ETL workflow is represented as a DAG,
- every task has associated a selectivity (defined as a ratio: output/input data volume),
- every task has associated a cost, which is a function of an input data size,
- a workflow is rearranged by means of operations: swap, factorize/distribute, merge/un-merge (as in [12]),
- a workflow rearrangement is guided by an optimization rule that moves (if possible) more selective tasks to the beginning of the workflow.

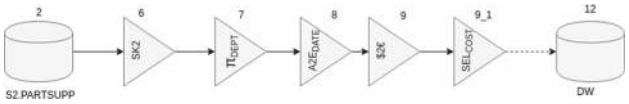

Fig. 12 Swappable and non-swappable tasks in an ETL workflow

[11] introduced a dependency graph—a structure that represents dependencies between tasks in a workflow. The graph is constructed by applying the 'swappability test,' proposed in [12]. Two given tasks are considered independent of each other if they are swappable, i.e., if they conform to the following four rules:

1. the tasks to swap must be adjacent to each other in the dependency graph,
2. the tasks to swap must have a single in/output schema and must have exactly one consumer of the output schema,
3. the tasks must have the same name for attributes in their in/output schema,
4. the tasks must generate the same schema before or after applying the *swap* operation.

As an example, let us consider a linear flow shown in Fig. 12. Recall that: (1) data source S2 stores costs and dates in the US format, (2) function $2€ (task 9) converts USD into EUR, and (3) task 9_1 selects some rows based on the value of attribute 'cost' (expressed in EUR). Since task 9_1 cannot be executed before task 9, they are non-swappable. Therefore, tasks 9 and 9_1 are dependent on each other. On the contrary, tasks 7 and 8 are swappable because they have non-intersecting schemas and they are independent on each other. Task 6 is also independent. Such checks are performed for each task in an ETL workflow, and the dependency graph is created.

The dependency graph is used for narrowing possible space of allowed rearrangements of tasks within a given workflow. To this end, the authors proposed a greedy heuristic that is applicable only to linear flows. For this reason, a given complex ETL workflow with multiple splits and merges (joins) is divided into n linear flows.

As an example, let us consider a complex ETL workflow, as shown in Fig. 13. The workflow is divided into the three following linear flows: LFI with tasks (3, 4, 5), LFII with tasks (6, 7, 8, 9, 10), and LFIII with tasks (12, 13).

Having divided a complex workflow into linear flows, each linear flow is optimized by rearranging its tasks. To this end, an algorithm was proposed whose intuition is as follows. Nodes of the dependency graph are ordered in a linear workflow by their selectivities. Less selective tasks are placed closer to the end of the flow (the target). This way, more selective tasks are moved toward the beginning of the flow (toward a data source). Tasks that depend on another tasks

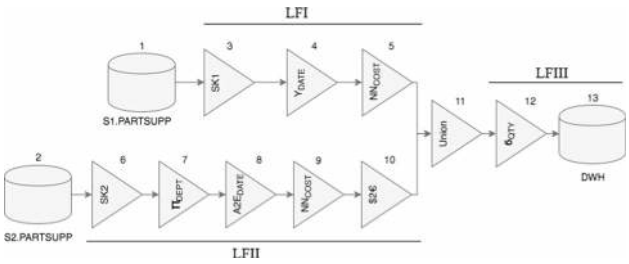

Fig. 13 Logical linear division of a complex ETL workflow

T must be placed to the right of T. This way, the dependencies between tasks represented in the dependency graph are respected.

Having optimized the linear flows, the final step is to combine the flows into larger linear flows that include all the tasks processing data from the source to the destination. For example, the linear flows from Fig. 13 are combined into two larger linear flows—the first one is composed of [LFI, LFIII] and the second one is composed of [LFII, LFIII].

Next, each combined linear flow is optimized by rearranging its tasks, as described above. There are tasks that may be moved from a given linear flow LF_m to the next linear flow LF_n, i.e., in the direction toward the end of a workflow. Such tasks are called forward transferable. There are also tasks that may be moved in the opposite direction, i.e., toward the beginning of a workflow. They are called backward transferable. An execution order within a combined linear flow is determined by the order implied by the dependency graph.

For example, in Fig. 13 tasks 5 and 9 have the same semantics (selecting rows with not null costs). Therefore, they can be moved forward to the beginning of linear flow III, such that the linear flow will be composed of tasks (5_9, 12, 13). Task 5_9 is a new task created by the factorize operator, having the same semantics as 5 and 9. Similarly, task 12 (selecting rows based on the value of attribute *QTY*) can be moved to LFI and LFII by applying the distribute operator.

Having constructed the combined linear flows, each combined flow is optimized by and algorithm whose intuition is as follows. All possible rearrangements of backward- and forward-transferable tasks are analyzed, and for each of them, an execution cost of the combined linear flow is computed. Next, the rearrangement with the lowest cost is selected.

6.3 Scheduling strategies for optimizing an ETL workflow

[9] propose a solution to optimize the performance of an offline batch ETL workflow in terms of execution time and memory consumption without the loss of data. To this end, a multi-threaded framework with incorporated ETL scheduler is presented, where each node of an ETL workflow is implemented as a thread. The proposed framework monitors, schedules, and guarantees the correct execution of an ETL workflow based on the proposed scheduling strategies such as 'minimum cost prediction (MCP),' 'minimum memory prediction (MMP),' and 'mixed policy (MxP).'

The MCP scheduling strategy is proposed to improve the performance of an ETL workflow by reducing its execution time. The ETL scheduler prioritizes activities to be scheduled at each step that have the largest volume of input data to process at that time. As a result, the activity with the largest input queue is able to process all data without any interruption from the ETL scheduler.

The MMP scheduling strategy is proposed to improve memory consumption by scheduling activities at each step that have the highest consumption rate (consumption rate = number of rows consumed/processing time of input data). As a result, the ETL scheduler maintains a data volume low in a system by scheduling the flow of activities whenever an input queue is exhausted due to higher memory consumption of the activity.

The MxP strategy is proposed to combine the benefits of MCP and MMP including operating system's default scheduling strategy (i.e., round-robin) by exploiting parallelism within an ETL workflow.

A set of scheduling policies is assessed for the execution of an ETL workflow. The results of incorporating scheduling policies are as follows: (1) MCP outperforms other scheduling strategies w.r.t. execution time, (2) MMP is better w.r.t. average memory consumption, and (3) MxP, which incorporates multiple scheduling strategies (e.g., MCP, MMP, or round-robin) by splitting an ETL workflow, achieves better time performance. The better time performance is achieved by either prioritizing memory intensive activities or by scheduling an ETL workflow to avoid blocking ETL operations.

6.4 Reusable patterns for optimizing an ETL workflow

[32] present reusable patterns, as a mean to characterize and standardize the representation of ETL activities along with the strategy to improve the efficiency an ETL workflow execution. The paper proposes to standardize the representation of frequently used ETL activities that involve a single transformation (e.g., surrogate key transformation, checking null values, and primary key violations), called *ETL Particles*. ETL activities that perform exactly one job and involve exactly one transformation (e.g., $2€ conversion) are called *ETL Atoms*. ETL Atoms that involve a linear flow of ETL particles are called *ETL Molecules*, and an ETL workflow is called *ETL Compound*.

The paper then presents a normal form of ETL activities, e.g., the normal form of activity 8 in Fig. 13 can be represented as follows:

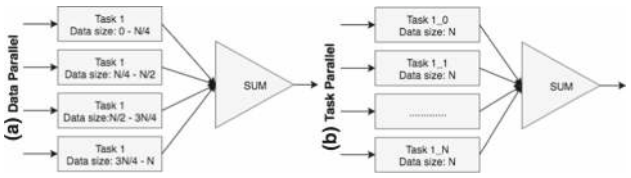

Fig. 14 Data parallelism and task parallelism

$$\boldsymbol{I}(\pi_{DEPT}), \boldsymbol{A2E}(DATE), \boldsymbol{O}(NN_{COST})$$

I is the input schema coming from activity π_{DEPT}. **A2E** is the specific template activity that converts the format of attribute DATE from American to European. **O** represents the output of activity 8 as an input to the next activity NN_{COST}. Similarly, activities 3 to 12 are ETL Molecules and the entire flow in Fig. 13 including activities and record sets represent the ETL Compound.

6.5 Parallelism for optimizing an ETL workflow

Besides the aforementioned optimization strategies, incorporating parallelism in an ETL workflow is another popular strategy to achieve better execution performance. Parallelism can be achieved either by partitioning the data into N subsets and process each subset in parallel sub-flows (data parallelism) or by using pipeline parallelism (task parallelism) as shown in Fig. 14.

For a simple ETL workflow, the data parallelism can work well. However, for a complex ETL workflow (e.g., nonlinear flows or flows with data and compute-intensive tasks), combination of data and task parallelism is required. Hence, a mixed parallelism is beneficial when either communication in a distributed environment is slow or the number of processors is large [33].

6.5.1 Parallelism in traditional data flow

Most of research on ETL workflow parallelism has focused on a traditional data flow parallelism, from which the most prominent one is the MapReduce framework [34]. It uses a generic key/value data model to process large-scale data in a parallel environment. MapReduce provides two functions, Map and Reduce, both having two input parameters, i.e., (1) a set of input data set in a key/value format and (2) an user-defined function (UDF). Map assigns a key/value pair to its input data using an UDF and produces a set of output key/value pairs. The Reduce function then groups the key/value pairs of its input data on the basis of keys, and finally, each group is processed using an UDF. Storage for MapReduce is based on distributed file system called Hadoop Distributed File System (HDFS) [35].

The MapReduce framework has its limitations, i.e., (1) it accepts a single set of input data set at a time in a key/value format, (2) it always executes in a strict order, i.e., first Map and then Reduce, (3) the output data from Map have to be stored into an intermediate file system, which makes data processing slow due to data partitioning and shuffling, and (4) developers have to write a custom code for the Map and the Reduce functions, which is hard to maintain and reuse.

Another parallel processing framework is PACT [36]. It is based on the Parallelization Contracts (PACTs), which consists of an Input Contract and an optional Output Contract. The Input Contract is the generalization of the Map and Reduce functions and takes an UDF and one or more data sets as an input. It also provides an extended set of Input Contracts i.e., 'Cross,' 'Math,' and 'CoGroup' functions, which complements Map and Reduce functionality and overcomes the limitations of MapReduce model, i.e., The Input Contract does not need to be executed in a fixed order and allows multiple inputs of key/value pairs. The Output Contract denotes different properties of the output data, which are relevant to parallelization. These properties can be either (1) preserving a partitioning property or (2) an ordering property on data that are generated by an UDF. An input UDF contains the strategy (such as partitioning, repartitioning, or broadcasting) for parallelizing in PACT.

For example, let us introduce another concept in our running example, REVIEWS (RID, PKEY, DEPT, REVIEW, SCORE, ENTRYDATE), which stores user reviews along with its computed score for each 'part' stored in a department. Now consider analytical query Q, which provides 'parts' from S1.PARTSUPP having COST greater than some amount, let us say 'c.' Query Q joins S1.PARTSUPP with REVIEWS, keeping only 'parts' where the review score is greater than some threshold 's,' and reduces the result set to the 'parts,' which are not present in S2.PARTSUPP for some department 'd.'

Figure 15 shows query Q as: (a) the MapReduce implementation and (b) the PACT implementation. The MapReduce implementation requires two stages of the MapReduce job. The first stage performs a join on the basis of PKEY in S1.PARTSUPP (S1) and REVIEWS (R) and carries out the specific selection based on S1. COST > 'c' and R.SCORE > 's.' The result from the first stage joins with the selection on S2.PARTSUPP (S2) on the basis of S2.DEPT = 'd' in the Map function of the second stage. The reducer in this stage performs an anti-join on the result set (when no S2.PARTSUPP row with an equal key is found).

The PACT implementation for the same query Q requires three separate user-defined functions (UDFs) attached to the Map contract to perform a selection on data sources S1, R, and S2 instead of a single user-defined function in a MapReduce scenario, such that each UDF is executed in parallel. The Match contract replaces the Reduce contract in the original MapReduce implementation. It matches the incoming key/value pairs from data sets with the same key and forms

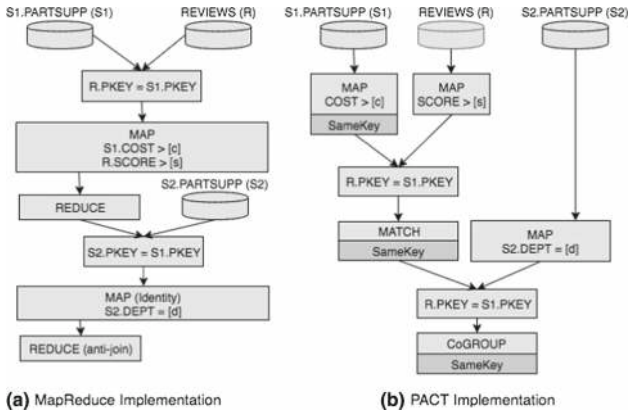

Fig. 15 MapReduce versus PACT implementation

an independent set based on the similar keys. The Match contract guarantees that the independent set of key/value pairs is supplied to exactly one parallel instance of the user-defined function so that each set is processed independently. Finally, the CoGroup contract implements the anti-join by releasing the rows coming from the Match contract. The CoGroup contract assigns the independent subsets with equal keys to the same group. The PACT implementation allows the flexibility to parallelize tasks by giving parallel hints using output contracts such as the SameKey contract. SameKey is the output contract attached to the Map, Match, and Co-Group contracts, which assures that the value and type of a key will not change even after applying the user-defied functions. Such hints are later exploited by an optimizer that generates parallel execution plans.

The strategy for the generation of an efficient parallel data flow is implemented in the Selinger-style SQL optimizer [37]. This kind of optimizer selects a globally efficient plan by generating multiple plans, starting from data sources, and pruning the costly ones based on partitioning and sort order properties. The plan with the lowest cost is selected as an optimized query plan. A PACT program is executed on a three-tier architecture [38] composed of: a PACT compiler, engine Nephele [39], and a distributed file system.

[40] propose a method to optimize a PACT program consisting of UDFs to process input data into multiple subsets, as follows:

1. use a static code analysis of an UDF (UDFs are considered as black-box activities with unknown semantics) to obtain relevant properties required to order UDFs;
2. enumerate all valid re-orderings for a given PACT program [41];
3. compute all possible alternative re-orderings using a cost-based optimizer to generate the execution plan by selecting execution strategies;
4. select and submit the plan with a minimum estimated cost for a parallel execution [36].

To overcome the aforementioned limitations of the MapReduce framework, an in-memory parallel computing framework Spark [42] uses multi-pass computation, i.e., computing components several times, using a Direct Acyclic Graph pattern. It also supports in-memory data sharing between multiple tasks. Spark allows the developers to create applications using API based on Resilient Distributed Dataset (RDD). RDD is a read-only multiset of data items distributed over a cluster of machines. RDD is placed on top of a distributed file system (typically HDFS) to provide multi-pass computations on data by rearranging the computations and optimizing data processing.

Another approach toward parallelizing a data flow is Structured Computations Optimized for Parallel Execution (SCOPE) [43]. It supports analyzing massive amount of data residing on clusters of hundreds or thousands of machines by means of and extensible scripting language similar to SQL. SCOPE uses a transformation-based optimizer which is a part of Microsoft's distributed computing platform called Cosmos. Cosmos accepts SCOPE scripts, translates them using SCOPE compiler, and finally invokes the SCOPE optimizer. The SCOPE optimizer introduces considerable parallelism based on a cost function. As presented in [44], the SCOPE optimizer generates a large number of execution plans by taking into account structural properties of data (e.g., partitioning, sorting, or grouping). The generated execution plans are then pruned using a cost model. However, this approach is restricted to relational operators (ROs) only, whereas in practice, it is important to optimize both ROs and UDFs.

The work presented in [45] acknowledges the importance of optimizing both ROs and UDFs. It extends the work presented in [44] by introducing parallelization techniques for UDFs. An UDF is treated as a black-box operation. In order to describe an UDF behavior and provide means for its parallelization, a set of UDF annotations were proposed. They describe pre- and post-conditions for partitioning and hints for an optimizer. For example, annotations BEGIN and END keywords, enclosed within annotations BEGIN PARALLEL and END PARALLEL mark the beginning and end point of the user-defined code. A script (with annotations) similar to SQLScript [46] is used to express complex data flows containing ROs and UDFs together. The main goal is to parallelize ROs and UDFs together, which is achieved by directly translating a RO into the internal representation of the proposed cost-based optimizer as described in [44] and by applying the 'Worker-Farm' pattern [47] on an UDF. A complete set of annotations is described in [45]. The proposed approach has two main limitations: (1) It supports UDFs implemented only as table functions and (2) it requires the ETL developer to annotate the custom code manually, i.e., in fact optimize the code manually.

6.5.2 Parallelism in an ETL workflow

Introducing parallelism into an ETL workflow is not a trivial task. The ETL developer has to decide which activities to parallelize, how much to parallelize, and when to parallelize before incorporating parallelism into an ETL workflow.

Task and code parallelism [48] propose a method to exploit parallelism at a code level by introducing strategies for both task and data parallelism. A set of constructs is proposed in order to enable the ETL developer to convert a linear ETL workflow into its corresponding parallel flow. The constructs are easy to use and do not require complicated modification of a non-parallel ETL workflow. However, the proposed solution is code based and requires the ETL developer to configure the degree of parallelization. Furthermore, this solution does not provide any cost model to estimate a performance gain.

Parallelizing by means of MapReduce [49,50] present a parallel dimensional ETL framework based on MapReduce called ETLMR. The focus of ETLMR is on star schema, snowflake schema, slowly changing dimensions (SCDs), and data-intensive dimensions. The ETLMR processes an ETL workflow in two stages. In the first stage, dimensions are processed using MapReduce tasks. In the second stage, facts are processed using another MapReduce task. Dimensions can be processed by using either of the following strategies: *One Dimension One Task* (ODOT) and *One Dimension All Tasks* (ODAT). In ODOT, dimensions are processed by the Map task using an UDF and then the processed data are propagated into a single Reduce task. In the Reduce task, user-defined transformations are performed on rows and then loaded into a DW. In ODAT, an output of the Map task is partitioned in a 'round-robin' fashion, i.e., the output is processed by all the Reduce tasks such that each Reduce task receives equal number of rows. The uniqueness of dimension key/values is maintained by using a global ID generator and a 'post-fixing' method, which merges rows with the same values but different keys into a single row. To optimize ODOT, keys with the same values are combined together in the Combiner task, to reduce the communication cost between the Map and Reduce tasks. Using the single Reduce task can become a bottleneck in case of a data intensive dimension; therefore, ODAT is used to overcome the bottleneck.

Fact processing in ETLMR requires looking up of dimension keys and aggregation (if required). If aggregations are not applicable, only the Map task is used and the Reduce task is dropped. Otherwise, only the Reduce task is used since aggregations must be completed from all the data in the Reduce task. Once the fact data are processed, it is loaded into a temporary buffer where it resides until the buffer is fully loaded. The Map and Reduce tasks then perform bulk loading in parallel into a DW.

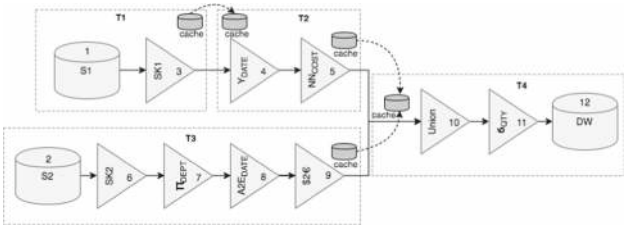

Fig. 16 ETL workflow partitioning

Partitioning and parallelization [51] propose ETL workflow partitioning and parallelization, as an optimization method. Vertical and horizontal partitioning is suggested. Vertical partitioning is impacted by tasks in an ETL workflow and the following tasks are distinguished: *row-synchronized*—it processes row by row (e.g., filter, lookup, split, data format conversion) and uses a shared cache to move data from task to task, *block*—processing cannot start until all rows are received by the task (e.g., aggregation), *semi-block*—receives rows from multiple tasks and merges them (e.g., join, set operators); processing of the task starts no sooner than all expected rows are received. The authors propose to partition and parallelize an ETL workflow at three levels. First, the whole ETL workflow is vertically partitioned into multiple sub-workflows—called execution trees. Second, execution trees are partitioned horizontally and each partition is run in parallel. Third, single tasks in an execution tree are parallelized by multi-threading.

Vertical partitioning is executed as follows. An ETL workflow analysis is run depth-first and it starts from a data source (the root of an ETL graph). All row-synchronized components (i.e., the ones that use a shared cache) are added into a new sub-workflow, so that their original order is preserved. If a block or semi-block task is found, then it becomes a root of a new sub-workflow.

Figure 16 explains the workflow partitioning method. Analyzing the ETL graph starts from data sources. For example, S1 and S2 create two separate execution trees T1 and T3, respectively. Task 3 is row-synchronized and it is added into T1, whereas row-synchronized tasks 6, 7, 8, and 9 are added into T3. Since task 4 is a block task, it becomes the root of a new execution tree T2 having activity 5 as its only child. Task 10 is a semi-block task, which forms execution tree T4, composed of tasks 11 and 12.

Once the execution trees are constructed, internal parallelization is carried out inside each of the execution trees. To this end, input data are partitioned horizontally into n disjoint partitions (n is parameterized), where each partition is processed by a separate thread. Finally, internal parallelization is carried out for tasks with a heavy computational load. To

find such a task, time is measured during which a task does not produce any output. If the time is greater than a given threshold, then the task becomes a candidate for internal parallelization. To parallelize a single task, multi-threading is applied, i.e., an input of the task is divided into n equal splits, each of which is run by a separate thread. Moving data between tasks within the same execution tree is implemented by means of a shared cache, whereas moving data between adjacent execution trees is implemented by means of coping data between separate cache (cf. dotted arrows in Fig. 16).

6.6 Quality metrics for ETL workflows

The goal of [23] is to reduce time and cost of ETL design, implementation, and maintenance (DIM) by incorporating some quality metrics into an ETL workflow design. A layered approach is proposed for an ETL DIM, where each layer represents a logical design, implementation, optimization, and maintenance. At each layer, some metrics are introduced (or refined from higher levels) that guide the ETL developer to produce a high-quality workflow. Furthermore, dependencies among metrics that impact DIM are identified and discussed. The following metrics are proposed in the so-called *QoX metric suite*: performance, recoverability, reliability, and maintainability—which characterize an ETL workflow as well as freshness—which characterizes processed data.

The aforementioned QoX metrics are interrelated and interdependent, which may lead to a contradictory behavior of a workflow. For example, on the one hand, increasing performance by partitioning and parallelization may increase freshness, but on the other hand, it decreases maintainability due to a more complex workflow design. The authors stress that some metrics can be used only at certain levels of DIM. For example, freshness and reliability can be handled at the physical level, but at the conceptual level their usefulness is questionable; conversely, performance is a pertinent metric at the conceptual, logical, and physical level of an ETL workflow design. The interdependencies between the quality metrics have been confirmed by some experiments. The authors report that: (1) increasing recoverability by means of recovery points decreases performance, as additional disk operations are required to store RPs and (2) the impact of some optimization techniques on the performance varies, e.g., increasing processing power does not improve performance linearly—there are parts for an ETL workload whose parallelization impacts performance stronger than the other parts.

6.7 Statistics for workflow optimization

Most of ETL workload optimization methods rely on various statistics. In [10], the authors provide a framework for gathering statistics for cost-based workload optimization. To this end, a workload must be divided into parts, called sub-expressions (SEs). The authors proposed to divide a workflow into SEs based on division points, which are ETL tasks (activities). The following tasks are used as division points: (1) materialization of intermediate results, (2) transformation of values of an attribute that is derived from the join of multiple relations and that is further used in another join, and (3) an UDF. Then, each SE is optimized independently. It must be stressed that the proposed framework does not deal with generating execution plans or estimating their costs, i.e., it is assumed that the set of SEs and their optimized execution plans exists and are delivered by an ETL optimizer module.

Finding an optimal execution plan is based on: (1) identifying different possible re-orderings of operators (tasks, activities) in a given SE and (2) estimating their execution costs, in the spirit of [11,12]. Each operator has a cost function that is based among others on: cardinalities of input relations (based on histograms), CPU and disk-access speeds, memory availability. The following operators are supported: select, project, join, group-by, and transform.

There are multiple sets of statistics (called candidate statistics set—CSS) suitable for optimizing a given SE. Some statistics can be computed from others, based on the following computation rules:

- the cardinality of the select operator can be estimated if the data distribution on a filtering attribute is known,
- for the project operator, output cardinalities and distributions are identical to the input ones,
- the cardinality of a join can be determined from the distributions of the input tables on a join attribute,
- the cardinality of group by is identical to the number of distinct values of grouping attributes,
- for the transform operator, output cardinalities and distributions are identical to the input ones,
- if there exists a histogram on any set of attributes of table T, then the cardinality of T can be computed by adding the values of buckets,
- if there exists a detailed histogram on attributes A and B, then a histogram for A can be computed by aggregating buckets on B.

Each CSS may have a different cost of collecting its statistics (e.g., CPU and memory usage). For this reason, a challenging task is to identify and generate a set of statistics to be collected by an ETL engine during its execution, such that: (1) the set of statistics can be used to estimate costs of all possible re-orderings of tasks within a given SE and (2) time and resource overhead of collecting the statistics is minimal. The authors identified that this is an NP-hard problem and to solve it they proposed a linear pro-

gramming algorithm. Finally, the authors suggested that the whole ETL optimization method is divided into the seven following steps: (1) identifying optimizable blocks by dividing a workflow into sub-expressions, (2) generating optimized sub-expressions by means of task reordering, (3) generating candidate statistics sets, (4) determining a minimal set of statistics, (5) augmenting an optimized SE by injecting into it a special component for collecting statistics, (6) running a SE and gathering statistics, and (7) optimizing the whole ETL workflow by means of cost-based techniques.

6.8 Commercial ETL tools

To the best of our knowledge, only two commercial ETL tools, namely IBM InfoSphere DataStage [52] and Informatica PowerCenter [53], provide some simple means of optimizing ETL workflows.

In IBM InfoSphere DataStage, the so-called *balanced optimization* is used. The optimization is included in the following design scenario: (1) an ETL workflow is designed manually (each task should be elementary, e.g., simple select from one table instead of a join), (2) the workflow is compiled into an internal representation, (3) optimization options are defined by the ETL developer, (4) the balanced optimization method is applied, and (5) the optimized workflow is produced to be run. The optimization process is guided by some parameters/options/hints including: (1) reduce a data volume retrieved from a data source (if possible), i.e., move data transformations, aggregations, sorting, duplicate removal, joins, and lookups into a data source, (2) alternatively, if possible, move processing into a data target, (3) use bulk loading to target, (4) maximize parallelism, and (5) use not more than a given number of nested joins.

Informatica PowerCenter implements the so-called *push-down optimization*. In this optimization, some ETL tasks that can be implemented as SQL commands are first identified. Second, these tasks are converted into SQL and executed either at an appropriate source or target database (depending on the semantics of the tasks). This approach leverages the processing power of a database in which data reside.

Other tools, including AbInitio, Microsoft SQL Server Integration Services, and Oracle Data Integrator, support only parallelization of ETL tasks, with a parameterized level of parallelism.

6.9 Summary

In this section, we discussed various approaches to the performance optimization of an ETL workflow, i.e., state-space search, dependency graphs, scheduling policies, and reusable patterns to optimize and ETL workflow execution. We also presented strategies that use parallelism in order to achieve execution performance in an ETL workflow. Finally, we presented optimization support available in commercial and open-source ETL tools. Below, we summarize the approaches on the basis of the following criteria:

1. *Autonomous Behavior*—whether an optimization method is automatic, semiautomatic (i.e., requires input from the ETL developer), or manual;
2. *UDF Optimization*—whether a method supports the optimization of user-defined functions;
3. *Monitoring*—whether a method or framework supports monitoring an ETL workflow in order to identify performance bottlenecks;
4. *Recommendation*—whether a method or framework provides recommendations to improve the implementation of an ETL workflow.

1. **State-space-based approaches** [10–12,30,31]

   Pros:
   - Optimization techniques based on execution costs and tasks reordering (similarly as query optimization techniques) are applied.
   - *Semi-autonomous behavior*—an input is an ETL workflow in the form of a graph. A given workflow is then transformed by an algorithm into a more efficient but semantically equivalent workflow.

   Cons:
   - *No UDF support*—the proposed optimization algorithms support only basic ETL operators.
   - *No monitoring and recommendation*—the proposed frameworks neither monitor ETL workflows for the performance bottlenecks nor provide hints on how to improve the performance of a workflow.
   - *Challenges*—one of the biggest challenges is the generation of an optimal ETL workflow using the proposed algorithms. If an ETL workflow is large and complex, the generation of an optimal workflow may take longer than the actual time of execution of the ETL workflow itself. Moreover, [12,31] are limited to a few transition techniques from one state to another and also do not give an account to translate an optimized logical model to its semantically equivalent physical implementation.

2. **Scheduling-based approach** [9]

   Pros:
   - Scheduling policies are applied to optimize an ETL workflow execution time and memory consumption.

- *Semi-autonomous behavior*—it is mainly focused on scheduling of ETL activities; the scheduling algorithm requires an ETL workflow as an input; the scheduling is based on predefined policies.
- *Monitoring*—it monitors the entire ETL workflow, but only for the purpose of scheduling ETL activities at the right time; it does not monitor the activities to find out performance bottlenecks.

Cons:

- *No UDF optimization and recommendation*—it does not specifically optimize the behavior of an UDF activity if it tends to be a bottleneck in an ETL workflow; the scheduling algorithm does not provide any recommendations to improve an input ETL workflow.
- *Challenges*—there is a possibility of losing data during scheduling; therefore, the approach will not be applicable to most of the traditional ETL processing.

3. **Parallelism-based approaches** [48–51]

Pros:

- *Semi-autonomous behavior*—the proposed approaches give a reasonable account for parallelizing the ETL activities, but the methods require a considerable amount of input from the ETL developer to decide which parts of an ETL workflow need to be parallelized, how much to parallelize, and where to put split points in order to enable parallelism.
- *UDF support*—the methods proposed in [49,50] support the ETL activities as UDFs but do not support their optimization.

Cons:

- *No monitoring and recommendation*—neither mechanisms for monitoring ETL workflows for bottlenecks nor recommendations to improve an input ETL workflow are supported.
- *Challenges*—the proposed solutions do not provide any cost model to identify the required degree of parallelism. Therefore, the ETL developer has to either perform trial and error method or execute the ETL transformations using test data to figure out the required degree of parallelism.

4. **QoX Suite** [23]

Pros:

- Analyzes various quality metrics and their inter dependencies for an ETL design (at a conceptual and logical level), implementation, optimization, and maintenance. The metrics are used for assessing the quality of a workflow.

Cons:

- *No autonomous behavior*—the approach provides only a theoretical framework.
- *UDF support*—not discussed.
- *No monitoring and recommendation*—neither methods for monitoring the performance of an ETL engine nor the functionality of improving an ETL design based on analyzing quality metrics has been discussed.
- *Challenges*—the biggest challenge is how to efficiently guide the ETL developer through subsequent stages of an ETL design, taking into account current values of the proposed metrics.

To conclude, the state-space search approaches [12,31] are considered among the first ones toward the logical optimization of an ETL workflow. [12] models the problem as a state-space search problem, where each state in a search space is a DAG. An optimal ETL workflow is achieved by choosing the optimal state from the number of generated states that are semantically equivalent to the original state. [31] focus on optimizing an ETL workflow for fault tolerance, performance, and freshness. These approaches served as the premise for the optimization of an ETL workflow and were later utilized by various researchers and specialists in this particular topic. [11] propose the dependency graph that is used for narrowing the space of allowed rearrangements of tasks within a given workflow. Most of ETL workload optimization methods presented in this survey rely on various statistics. In [10], the authors provide a framework for gathering statistics for cost-based workload optimization.

The second group of approaches focuses on strategies that use parallelism as a mean for increasing ETL execution performance, but most of them focus on data flow parallelism. [48] introduce a method to parallelize an ETL workflow (developed in some programming language) by introducing both task parallel and data parallel strategies. [49,50] present an ETL framework based on MapReduce. Although [48–51] give a reasonable account for parallelization, these methods require a considerable amount of input from the ETL developer to decide which parts of an ETL workflow need to be parallelized, how much to parallelize, and where to put the split points in order to enable parallelism.

7 Conclusions

An ETL workflow comprises numerous activities, e.g., data extraction, validation, transformation, cleaning, conversion,

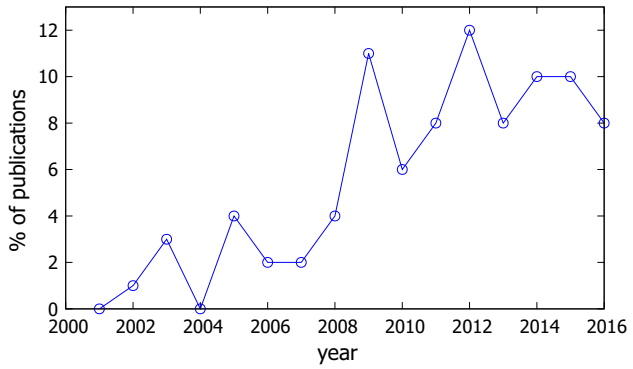

Fig. 17 The distribution of ETL publications (in % of total 229 since 2000 (based on DBLP)

deduplication, and loading. All these activities can be represented and executed in many distinctive ways, e.g., using relational operators or user-defined functions (implemented in various programming languages), which results in a complex design of an ETL workflow. Designing an error-free and efficient ETL workflow is a complex and expensive task. For this reason, the research community has contributed a significant amount methods for the design, development, deployment, and optimization of an ETL workflow.

The most intensive research on ETL took place during the last 10 years. Figure 17 shows the distribution of publications on ETL (expressed in % of total 229 publications), listed on DBLP since 2001. With the growing need to warehouse and analyze Big Data, we may expect that the interest of the community on ETL will even increase in the nearest future. The still open issues on ETL development, discussed in this survey, become much more difficult to solve in the area of Big Data. In this paper, we have presented the state of the art and current trends in the whole process of an ETL workflow development and optimization.

7.1 ETL workflow development: summary

There exist multiple techniques for an ETL workflow development. At the conceptual level, there are methods involving graphs, semantic Web ontology, UML notation, and BPMN. All these methods propose diverse approaches to design an ETL conceptual model in a precise and productive way. Then, at a logical level, there exist approaches using graphs, which supplement a generic behavior of a logical design. Additionally, there are methods to incorporate quality metrics in an ETL workflow for its efficient and reliable execution. For a physical implementation of a logical design of an ETL workflow, there exist methods that give detailed account on translating a logical model into its corresponding physical implementation.

Based on the analysis of the presented methods for each development stage, i.e., conceptual modeling, logical modeling, and physical implementation, we draw the following conclusions.

1. There are diverse methods for constructing a conceptual model of an ETL workflow, e.g., graph-based [7,8], UML-based [14], ontology-based [15,17,18,24,54], and BPMN [19–22], from which we can conclude that the research community has not yet concurred upon the standard notation and model for representing an ETL conceptual design. As a consequence, it is difficult to develop guidelines for validating an ETL design.
2. The discussed graph-based [7,8] and UML-based [14] methods require the ETL developer to extensively provide input during the modeling and design of an ETL workflow; thus, it can be error-prone, time-consuming, and inefficient.
3. Most of the methods are designed only for structured data [7,8,14,19,22], and the support for semi-structured and unstructured data is very limited. Since the variety of data forMats is growing rapidly and most of data are unstructured, it is important to extend the support for unstructured data in an ETL workflow.
4. Almost all of the discussed design methods are based on the traditional ETL operators (e.g., join, union, sort, aggregate, lookup, and convert), and they mostly do not consider UDFs as ETL activities.
5. Few methods [22,31] put emphasis on the issues of efficient, reliable, and improved execution of an ETL workflow using quality metrics, as described in [23]. However, most of the methods in practice do not capture or track these quality metrics.

7.2 ETL workflow optimization: summary

We also discussed methods for the optimization of an ETL workflow and afterward narrowed down our point of interest to parallelization techniques. The following ETL optimization approaches have been proposed so far: state-space search, dependency graph, and scheduling policies. The state-space search approach serves as the foundation for the optimization of an ETL workflow for other research. The dependency graph approach focuses on the optimization of a linear and a nonlinear logical ETL workflow. The scheduling policies are proposed to optimize the ETL workflow with respect to execution time and memory consumption.

In the literature, there exist multiple methods that revolve around data flow parallelism [34,36,40,43–45]. However, research on an ETL workflow parallelism has not appealed much consideration.

Based on the analysis of the presented methods for and ETL workflow optimization, we draw the following conclusions.

1. [9,11,12,30–32,48–51] require extensive amount of input from the ETL developer to optimize an ETL workflow, which makes his/her job very complicated and time-consuming. Furthermore, the proposed methods require the ETL developer to be highly technical in programming as well as cautious in order to understand the quality metrics and their impact on the performance.
2. The methods do not consider the optimization of UDFs in a comprehensive manner. As UDFs are commonly used in an ETL workflow to overcome the limitations of traditional ETL operators, it is important to optimize UDFs along with traditional ETL operators. Since an UDF is typically considered as a black-box activity and its semantics is unknown, it is very difficult to optimize its execution.
3. Currently, there is no such framework that autonomously monitors an ETL workflow to find out which ETL activities hinder its performance and gives recommendations to the ETL developer how to increase its performance.

7.3 Open Issues

On the basis of this literature review, we can conclude our paper with the following open issues.

1. There is a need for a unified model for an ETL workflow, so that it is easier to validate an ETL design for its quality metrics.
2. There is a need to consolidate and fully support UDFs in an ETL workflow along with traditional ETL operators.
3. There is a need for an ETL framework that shall reduce the work of the ETL developer from a design and performance optimization perspective. The framework should provide recommendations on: (1) an efficient design an ETL workflow according to the business requirements and (2) how and when to improve the performance of an ETL workflow without conceding other quality metrics.
4. To improve the execution performance of an entire ETL workflow, techniques based on task parallelism, data parallelism, and a combination of both for traditional ETL operators as well as UDFs are required.

A new and yet almost unexplored area is handling structural changes in data sources at an ETL layer. In practice, data sources change their structures (schemas) frequently. Typically after such changes, an ETL workflow cannot be executed and must be repaired (cf. the maintainability metric in Sect. 6.6). Such a repair is done manually by the ETL designer, as neither of commercial and open-source ETL tools supports (semi-)automatic repairs of ETL workflows. The tools support only impact analysis.

So far, only two research approaches have been proposed that address this problem, namely *Hecataeus* [55] and *E-ETL* [56]. In [55], an ETL workflow is manually annotated with rules that define the behavior of the workflow in response to a data source change. In [56], a case-based reasoning is used to semiautomatically (or if possible—automatically) repair an ETL workflow. Since both of the approaches do not provide comprehensive solutions, this problem still needs substantial research.

A rapidly growing need for analyzing Big Data calls for novel architectures for warehousing the data, such as a *data lake* [57] or a *polystore* [58]. In both of the architectures, ETL processes serve similar purposes as in traditional DW architectures. Since Big Data exist in a multitude of formats and the relationships between data often are very complex, ETL workflows are much complex than in traditional DW architectures. Such architectures also need data transformations and cleaning (often on-the-fly, i.e., when a query is executed). For these reasons, designing ETL workflows for Big Data challenging. Multiple off-the-box ETL tasks are not suitable for processing Big Data, and such tasks have to be implemented by UDFs. Since a Big Data ETL engine processes much complex ETL workflows and much larger data volumes, the performance of the engine becomes vital.

The consequence of the aforementioned observation is that designing and optimizing ETL workflows for Big Data is much more difficult than for traditional data. Therefore, the open issues identified for traditional ETL workflows become even more difficult to solve in the context of Big Data.

7.4 Extendible theoretical ETL Framework

Based on the open issues identified in this survey, we present a theoretical *ETL Framework*. It can be extended with various components, and some of them offer functionality that is not supported in the current state-of-the-art solutions. An architecture of the *ETL Framework* is shown in Fig. 18.

ETL Workflow Designer represents any standard open-source or commercial ETL tool for designing an ETL workflow. The middle layer is extendible and consists of the four following components:

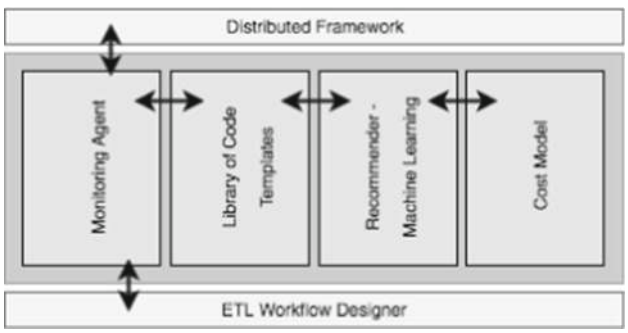

Fig. 18 The overall architecture of the *ETL Framework*

– A *Monitoring Agent*—it allows to: (1) monitor ETL workflow executions, (2) identify performance bottlenecks, (3) report errors, (4) schedule executions, and (5) gather various performance statistics. This is a standard component of any ETL engine.
– A *Library of Code Templates*—it contains the set of code templates that allow the ETL developer to write parallel UDFs to be executed in a distributed environment, e.g., a template for MapReduce. The ETL developer only has to provide the Map and Reduce functions, whereas MapReduce configurations (i.e., partitioning parameters, number of nodes) will be provided by the *ETL Framework*, which actually are very critical to achieve right degree of parallelism.
– A *Recommender*—it includes a set of machine learning algorithms to optimize a given ETL workflow (based on metadata collected during past ETL executions) and to generate a more efficient version of the workflow. Since there are a few algorithms that can be applied to optimizing a workflow, the ETL developer should be able to experiment with alternative algorithms and compare their optimization outcomes. To this end, the *Recommender* module should allow to plug in various optimization algorithms.
– A *Cost Model*—the algorithms used by the *Recommender* need cost models. Multiple cost models have been proposed in the research literature, and the module should allow to extend the set of cost models applied to the optimization.

The top layer in the architecture is the *Distributed Framework*. Its task is to execute parallel codes of UDFs (developed using the code templates) in a distributed environment, in order to improve the overall execution performance of an ETL workflow.

7.5 Future work: our approach to ETL optimization

Currently we are implementing the following two main components of the *ETL Framework*, as first steps toward building the complete *Framework*: (1) a *Configurable–parallelizable UDF Component* (*cp-UDF*)—to provide the library of reusable parallel algorithmic skeletons for the ETL developer and (2) a *cost model*—to generate the most efficient execution plan for an ETL workflow.

cp-UDF exposes a library of reusable parallel algorithmic skeletons (e.g., MapReduce Paradigm Skeleton, Spark, Flink) for the ETL developer, in order to support parallel implementations of UDFs. The skeletons contain the main configurations and necessary functions for a relevant distributed system and would only require an ETL developer to write the main functionality of an UDF. For example, in case of the MapReduce skeleton, *cp-UDF* requires only the Map function and Reduce function to be implemented. The performance tuning and configuration, runner class as well as input and output configuration parameters will be provided by *cp-UDF* itself.

Having provided a skeleton augmented with a user code, *cp-UDF* generates multiple parallel variants of the code. For example, in case of MapReduce, *cp-UDF* will generate a parallel skeleton with different configurations of a virtual machine (VM), where a given program is to be executed, i.e., the same program can be executed on multiple different VMs that may result in different execution time and monetary cost (in case of cloud distributed frameworks).

The *cost model* finds a sub-optimal optimized execution plan for an ETL workflow, based on execution time and monetary cost constraints provided by the ETL developer. Assuming that: (1) an ETL workflow consists of n different computationally intensive UDFs and (2) *cp-UDF* may generate m parallel variants of each UDF, there are m^n combinations of code variants. Finding an optimal code variant may be mapped to Multiple Choice Knapsack Problem (MCKP) [59]. Currently, we are implementing the *cost model* component and are applying dynamic programming for solving MCKP.

Acknowledgements The research of F. Ali has been funded by the European Commission through the Erasmus Mundus Joint Doctorate 'Information Technologies for Business Intelligence Doctoral College' (IT4BI-DC). The research of R. Wrembel has been funded by the National Science Center Grant No. 2015/19/B/ST6/02637.

References

1. Jensen, C.S., Pedersen, T.B., Thomsen, C.: Synthesis Lectures on Data Management. Morgan & Claypool Publishers, San Rafael (2010)
2. Andzic, J., Fiore, V., Sisto, L.: Extraction, transformation, and loading processes. In: Wrembel, R., Koncilia, C. (eds.) Data Warehouses and OLAP: Concepts, Architectures and Solutions. Idea Group Inc. (2007). ISBN 1-59904-364-5
3. Patil, P., Rao, S., Patil, S.: Data integration problem of structural and semantic heterogeneity: data warehousing framework models for the optimization of the ETL processes. In: Proceedings of ACM International Conference and Workshop on Emerging Trends in Technology (2011)
4. Gartner magic quadrant for data integration tools (2017)
5. 10 open source ETL tools. Data science central. www.datasciencecentral.com/profiles/blogs/10-open-source-etl-tools. Accessed 10 June 2017 (2015)
6. Awad, M.M., Abdullah, M.S., Ali, A.B.M.: Extending ETL framework using service oriented architecture. Proc. Comput. Sci. **3**, 110–114 (2011)

7. Vassiliadis, P., Simitsis, A., Skiadopoulos, S.: Conceptual modeling for ETL processes. In: Proceedings of ACM International Workshop on Data Warehousing and OLAP (DOLAP) (2002)
8. Simitsis, A., Vassiliadis, P.: A methodology for the conceptual modeling of ETL processes. In: Proceedings of the of Conference on Advanced Information Systems Engineering (CAiSE) (2003)
9. Karagiannis, A., Vassiliadis, P., Simitsis, A.: Scheduling strategies for efficient ETL execution. Inf. Syst. **38**(6), 927–945 (2013)
10. Halasipuram, R., Deshpande, P.M., Padmanabhan, S.: Determining essential statistics for cost based optimization of an ETL workflow. In: Proceedings of International Conference on Extending Database Technology (EDBT), pp. 307–318 (2014)
11. Kumar, N., Kumar, P.S.: An efficient heuristic for logical optimization of ETL workflows. In: Proceedings of International Conference on Very Large Data Bases (VLDB). Springer, Berlin (2011)
12. Simitsis, A., Vassiliadis, P., Sellis, T.: State-space optimization of ETL workflows. IEEE Trans. Knowl. Data Eng. (TKDE) **17**(10), 1404–1419 (2005)
13. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Pearson Higher Education, London (2004)
14. Trujillo, J., Luján-Mora, S.: A UML based approach for modeling ETL processes in data warehouses. In: Proceedings of International Conference on Conceptual Modeling (ER). Springer, Berlin (2003)
15. Skoutas, D., Simitsis, A.: Designing ETL processes using semantic web technologies. In: Proceedings of ACM International Workshop on Data Warehousing and OLAP (DOLAP) (2006)
16. McGuinness, D.L., Van Harmelen, F., et al.: OWL web ontology language overview. W3C recommendation. https://www.w3.org/TR/owl-features/. Accessed 05 June 2017 (2004)
17. Skoutas, D., Simitsis, A.: Ontology-based conceptual design of ETL processes for both structured and semi-structured data. Int. J. Semant. Web Inf. Syst. (IJSWIS) **3**(4), 1–24 (2007)
18. Skoutas, D., Simitsis, A., Sellis, T.: Ontology-driven conceptual design of ETL processes using graph transformations. In: Journal on Data Semantics XIII. Lecture Notes in Computer Science, pp. 120–146. Springer, Berlin (2009)
19. El Akkaoui, Z., Zimányi, E.: Defining ETL workflows using BPMN and BPEL. In: Proceedings of ACM International Workshop on Data Warehousing and OLAP (DOLAP) (2009)
20. El Akkaoui, Z., Zimànyi, E., Mazón, J.N., Trujillo, J.: A model-driven framework for ETL process development. In: Proceedings of the ACM International Workshop on Data Warehousing and OLAP (DOLAP) (2011)
21. Oliveira, B., Belo, O.: BPMN patterns for ETL conceptual modelling and validation. In: Foundations of Intelligent Systems. Springer, Berlin (2012)
22. Wilkinson, K., Simitsis, A., Castellanos, M., Dayal, U.: Leveraging business process models for ETL design. In: Proceedings of the International Conference on Conceptual Modeling (ER). Springer, Berlin (2010)
23. Simitsis, A., Wilkinson, K., Castellanos, M., Dayal, U.: QoX-driven ETL design: reducing the cost of ETL consulting engagements. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (2009)
24. Simitsis, A., Skoutas, D., Castellanos, M.: Representation of conceptual ETL designs in natural language using semantic web technology. Data Knowl. Eng. (DKE) **69**(1), 96–115 (2010)
25. Vassiliadis, P., Simitsis, A., Skiadopoulos, S.: Modeling ETL activities as graphs. In: Proceedings of International Workshop on Design and Management of Data Warehouses (DMDW) (2002)
26. Simitsis, A., Vassiliadis, P., Sellis, T.: Logical optimization of ETL workflows. In: Proceedings of Hellenic Data Management Symposium. Citeseer (2005)
27. Vassiliadis, P., Simitsis, A., Georgantas, P., Terrovitis, M., Skiadopoulos, S.: A generic and customizable framework for the design of ETL scenarios. Inf. Syst. **30**(7), 492–525 (2005)
28. Simitsis, A., Vassiliadis, P.: A method for the mapping of conceptual designs to logical blueprints for ETL processes. Decis. Support Syst. **45**(1), 22–40 (2008)
29. Vassiliadis, P., Simitsis, A., Georgantas, P., Terrovitis, M.: A framework for the design of ETL scenarios. In: Proceedings of the Conference on Advanced Information Systems Engineering (CAiSE). Springer, Berlin (2003)
30. Tziovara, V., Vassiliadis, P., Simitsis, A.: Deciding the physical implementation of ETL workflows. In: Proceedings of ACM International Workshop on Data Warehousing and OLAP (DOLAP) (2007)
31. Simitsis, A., Wilkinson, K., Dayal, U., Castellanos, M.: Optimizing ETL workflows for fault-tolerance. In: Proceedings of IEEE International Conference on Data Engineering (ICDE) (2010)
32. Vassiliadis, P., Simitsis, A., Baikousi, E.: A taxonomy of ETL activities. In: Proceedings of ACM International Workshop on Data Warehousing and OLAP (DOLAP) (2009)
33. Chakrabarti, S., Demmel, J., Yelick, K.: Modeling the benefits of mixed data and task parallelism. In: Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures (1995)
34. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
35. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. IEEE Mass Storage Syst. Technol. (MSST) (2010)
36. Battré, D., Ewen, S., Hueske, F., Kao, O., Markl, V., Warneke, D.: Nephele/pacts: a programming model and execution framework for web-scale analytical processing. In: Proceedings of ACM Symposium on Cloud Computing. ACM (2010)
37. 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 ACM SIGMOD International Conference on Management of Data (1979)
38. Alexandrov, A., Heimel, M., Markl, V., Battré, D., Hueske, F., Nijkamp, E., Ewen, S., Kao, O., Warneke, D.: Massively parallel data analysis with pacts on nephele. Proceedings of International Conference on Very Large Data Bases (VLDB) (2010)
39. Warneke, D., Kao, O.: Nephele: efficient parallel data processing in the cloud. In: Proceedings of ACM International Workshop on Many-Task Computing on Grids and Supercomputers (2009)
40. Hueske, F., Peters, M., Krettek, A., Ringwald, M., Tzoumas, K., Markl, V., Freytag, J.: Peeking into the optimization of data flow programs with mapreduce-style UDFS. In: Proceedings of IEEE International Conference on Data Engineering (ICDE) (2013)
41. Hueske, F., Peters, M., Sax, M.J., Rheinländer, A., Bergmann, R., Krettek, A., Tzoumas, K.: Opening the black boxes in data flow optimization. In: Proceedings of International Conference on Very Large Data Bases (VLDB) (2012)
42. Apache Spark—lightning-fast cluster computing. http://spark.apache.org/. Accessed on 22 July 2016
43. Chaiken, R., Jenkins, B., Larson, P.Å., Ramsey, B., Shakib, D., Weaver, S., Zhou, J.: Scope: easy and efficient parallel processing of massive data sets. In: Proceedings of International Conference on Very Large Data Bases (VLDB) (2008)
44. Zhou, J., Larson, P.A., Chaiken, R.: Incorporating partitioning and parallel plans into the scope optimizer. In: Proceedings of IEEE International Conference on Data Engineering (ICDE) (2010)
45. Große, P., May, N., Lehner, W.: A study of partitioning and parallel UDF execution with the SAP HANA database. In: Proceedings of ACM International Conference on Scientific and Statistical Database Management (SSDBM) (2014)

46. Binnig, C., May, N., Mindnich, T.: SQLScript: Efficiently Analyzing Big Enterprise Data in SAP HANA. In: Datenbanksysteme für Business, Technologie und Web (BTW) (2013)
47. Große, P., Lehner, W., May, N.: Advanced analytics with the SAP HANA database. In: DATA (2013)
48. Thomsen, C., Pedersen, T.B.: Easy and effective parallel programmable ETL. In: Proceedings of ACM International Workshop on Data Warehousing and OLAP (DOLAP) (2011)
49. Liu, X., Thomsen, C., Pedersen, T.B.: ETLMR: a highly scalable dimensional ETL framework based on mapreduce. In: Proceedings of International Conference on Data Warehousing and Knowledge Discovery (DaWaK). Springer, Berlin (2011)
50. Liu, X., Thomsen, C., Pedersen, T.B.: ETLMR: a highly scalable dimensional ETL framework based on mapreduce. In: Transactions on Large-Scale Data and Knowledge-Centered Systems, LNCS. Springer, Berlin (2013)
51. Liu, X., Iftikhar, N.: An ETL optimization framework using partitioning and parallelization. In: Proceedings of the 30th Annual Symposium on Applied Computing. ACM (2015)
52. Lella, R.: Optimizing BDFS jobs using InfoSphere DataStage Balanced Optimization. IBM Developer Works (2014)
53. How to Achieve Flexible, Cost-effective Scalability and Performance through Pushdown Processing. Informatica whitepaper. https://www.informatica.com/downloads/pushdown_wp_6650_web.pdf. Accessed 01 June 2017 (2007)
54. Bergamaschi, S., Guerra, F., Orsini, M., Sartori, C., Vincini, M.: A semantic approach to ETL technologies. Data Knowl. Eng. (DKE) **70**(8), 717–731 (2011)
55. Manousis, P., Vassiliadis, P., Papastefanatos, G.: Impact analysis and policy-conforming rewriting of evolving data-intensive ecosystems. J. Data Semant. (2015). doi:10.1007/s13740-015-0050-3
56. Wojciechowski, A.: ETL workflow reparation by means of case-based reasoning. Inf. Syst. Front. (2017). doi:10.1007/s10796-016-9732-0
57. Terrizzano, I., Schwarz, P., Roth, M., Colino, J.E.: Data wrangling: the challenging journey from the Wild to the lake. In: Proceedings of Conference on Innovative Data Systems Research (CIDR) (2015)
58. Duggan, J., Elmore, A.J., Stonebraker, M., Balazinska, M., Howe, B., Kepner, J., Madden, S., Maier, D., Mattson, T., Zdonik, S.: The BigDAWG Polystore system. SIGMOD Rec. **44**(2), 11–16 (2015)
59. Ibaraki, T., Hasegawa, T., Teranaka, K., Iwase, J.: The multiple choice knapsack problem. J. Oper. Res. Soc. Jpn. **21**(1), 59–93 (1978)