On Three Missing Pieces in the Data Integration Puzzle (Position Paper)

Robert Wrembel[1]

[1]Poznan University of Technology, Poznań, Poland

Abstract

Data integration (DI) for years has been among the most frequently researched topics. A common goal of DI is to make heterogeneous and distributed data available for an end user in a unified format that is suitable for analyses. Research and development works resulted in a few standard DI architectures. In all of them data are moved from data sources (DSs) into an integrated system by means of an integration layer. This layer runs DI processes that are complex workflows composed of multiple tasks responsible for extracting data from DSs and making them available for analytics. Methods for developing DI processes have been researched and developed for decades, but designing and managing DI processes is still difficult and time costly. Moreover, the still open problems include: (1) the optimization of DI processes, (2) managing user-defined functions in DI processes, especially available as black-boxes, and (3) discovering and managing broken data lineage in a DI pipeline. In this position paper we formulate the three research hypotheses concerning the aforementioned problems and show how these hypotheses can be verified to develop the missing solutions.

Keywords

data integration, data integration processes, user-defined functions, data lineage

1. Introduction and motivation

For years, the widespread of complex, data-driven systems has been observed (e.g., in medicine, agriculture, smart cities). They produce huge volumes of highly heterogeneous data (a.k.a. big data). These data need to be integrated to feed various analytical and machine learning (ML) applications. Consequently, *data integration* (DI) techniques are core components of information systems. *DI architectures and processes* are among very frequently researched topics [1, 2, 3, 4].

DI aims at consolidating data from multiple distributed and heterogeneous data sources (DSs), to deliver these data to an end user in a common format. Research and development works resulted in a few standard DI architectures, namely: federated [5] and mediated [6], data warehouse (DW) [7], data lake (DL) [8], data lake house (DLH) [9], and (7) data mesh [10]. In all of these architectures, data are moved from DSs into an integrated system by means of an integration layer. This layer is implemented by a sophisticated software, which runs the so-called DI processes.

DI processes range from simple to complex workflows of dozens to thousands of tasks. These tasks are responsible for extracting data from DSs, transforming data into a common model and data structures, cleaning data, and loading them into either a central repository (i.e., a DW, DL, or DLH) or making them available in virtual integration architectures (i.e., federated, mediated, or data mesh). DI processes are managed by a dedicated software, called a DI engine (ETL engine in DW architectures) [11, 12, 13].

Methods for designing DI processes have been researched and developed for decades (see [14, 15]). Despite these substantial number of works, designing and managing DI processes is still difficult and time costly [16]. Moreover, the still **open fundamental research problems** include: (1) the optimization of DI processes, (2) managing user-defined functions (UFDs) in DI processes, and (3) managing data lineage in a DI pipeline. In this position paper we discuss possible solutions to these problems. This discussion is motivated by our cooperation with the IT sector and with the banking sector.

2. Related research

2.1. Optimization of DI processes

Any DI process has to finish its work within a given time window, typically a few hours, in order to make a DW/DL/DLH available for users. Because a DI process (1) moves large volumes of heterogeneous data between DSs and a DW/DL/DLH and (2) executes complex data processing algorithms, its execution is time costly. Therefore, providing solutions for efficient execution of DI processes (a.k.a. DI process optimization) is of high importance, since it is crucial for the whole DI architecture. Big data (with their volume and complexity) further increase the difficulty of DI process optimization. In order to reduce the execution time of a DI process, a few classes of solutions have been proposed by industry and research.

Industry approaches are based on the following op-

Published in the Proceedings of the Workshops of the EDBT/ICDT 2025 Joint Conference (March 25-28, 2025), Barcelona, Spain

robert.wrembel@cs.put.poznan.pl (R. Wrembel)

0000-0001-6037-5718 (R. Wrembel)

timization techniques: scaling-up or scaling-out a data integration server or adding a specialized hardware like FPGAs [17, 18], parallel processing of DI tasks, and moving the execution of some DI tasks into a DS - this technique is commonly called push-down [19, 20].

Research approaches draw upon the following ideas. The first class of solutions is based on **changing the order of DI tasks** (a.k.a. task reordering), so that a new reordered process is more efficient that the original one [21, 22, 23, 24]. These methods are computationally too expensive (exponential) [21] - the search space of all possible valid orders of tasks in such processes is too large to be fully searched. As a consequence, the applicability of these methods is limited to rather simple DI processes. Moreover, none of the aforementioned approaches supports the optimization of DI processes with UDFs. The reordering of operators is based on the semantics of the operators, which is well known and understood for traditional relational operators. On the contrary, the semantics of UDFs is frequently unknown.

A special class of reordering is the **push-down** technique. The principle of this technique is to move data processing from a DI engine into a DS server for execution [19, 20, 25]. However, push-down so far has been researched and developed mainly for relational DSs. To the best of our knowledge, push-down for a few non-relational DSs is available in *Informatica* and *Spark*. *Informatica* is able to push-down filters, joins, and aggregations into a *Hadoop* cluster. *Spark* query programming library allows to push-down filter predicates and aggregations to Parquet, ORC, and a few relational databases connected to via a JDBC driver [26].

Despite these solutions are available in software tools, still open problems concern among others: (1) efficient implementation of a pushed-down operation in a DS, given its functionality and internal features, (2) an overall method for assessing profitability of push-down, (3) extending push-down to key-value, column-family, document, and graph storage.

The second class of solutions uses **parallel processing** of either individual tasks within a process or portions of the process [27, 28, 29, 30, 31]. A challenge in this approach is to figure out the most efficient parallelization scheme for a given DI task, or a subset of tasks, or the whole DI process [32], especially when DI processes use UDFs. Task parallelization has been researched quite extensively and a few sound solutions to this challenge have been proposed.

2.2. UDFs in DI processes

Big data integration processes use not only predefined DI tasks available in DI design, development, and management tools [12], but also require the deployment of **user defined functions**. UDFs allow to implement code snippets of a logic tailored to a specific and non-typical, data processing problem. Thus, UDFs extend the functionality of a DI architecture with tailored tasks. UDFs represent code written in multiple programming languages, external to a DI design environment and a DI engine. Typically, such UDFs are treated by a DI engine as black-boxes (further called black-box UDFs - BBUDFs), i.e., code snippets whose internal logic and performance characteristics are unknown.

Multiple works addressed the usage of UDFs in data engineering. They can be categorized as: (1) explicit code annotations by a programmer [33, 34, 35, 36], (2) static code analysis and debugging [33, 37, 38], (3) efficient compilation and execution [39, 40, 41, 42], (4) UDFs in database queries [43, 44, 45, 46], and (5) UDFs in data pipelines [36, 47, 48]. Most of these solutions assume that UDFs are treated as white-boxes (their implementation code is accessible), which prevents their applications to BBUDFs.

Using a BBUDF in a DI process makes such a process difficult to optimize and manage (e.g., impact analysis). First, because the internal logic of such a BBUDF is unknown. Second, its performance characteristics have to be learned, which is a time consuming task. Third, it causes data lineage (see Section 2.3) extremely challenging. For this reason, discovering the internal logic and performance characteristics of BBUDFs is of paramount importance for building efficient DI processes with data lineage.

2.3. Data lineage in DI processes

Data lineage is a set of techniques that document the lifecycle of data, including source information and any data transformations that have been applied within any DI process [49]. These techniques are crucial for data management, governance, and compliance, since they provide insights into where data came from and how they were processed along a data pipeline. Data lineage techniques have been researched for decades. They can be categorized according to a few taxonomies.

The first one divides them into annotation-based and inversion-based. In the **annotation-based** techniques, input data are annotated with information about their origin and how they were processed along a data pipeline. The annotations are propagated throughout the whole pipeline to be included in final results [50, 51, 52, 53, 54]. The **inversion-based** techniques invert queries, trying to infer the original data that produced the result of a given query [55, 56, 57]. To this end, the so-called inversion functions are used.

The second taxonomy divides lineage techniques into tuple-level and value-level. The **tuple-level** lineage supports tracking the processing of individual data rows, e.g., [58, 55]. The **value-level** lineage supports tracking the

lineage of specific values within rows, e.g., [56].

The third taxonomy organizes lineage into eager and lazy. In the **eager** technique, a lineage information is built for all output data while executing a query (analysis) [59, 60, 61, 62, 63, 64]. In the **lazy** technique, this information is constructed after a query was executed, e.g., [55].

The lineage information is typically represented either by IDs that are attached to source tuples (and in some cases their individual attributes), e.g., [59, 61, 62] or by additional structures that relate source and final (after processing) data, e.g., [64].

More advanced solutions using annotation were presented in [53, 65, 66] - these solutions are known as the **semiring** provenance model. It leverages the mathematical structure of semirings to represent and analyze a lineage information.

3. The missing pieces

From the analysis of the state of the art on managing DI processes, we draw the following conclusions. First, while the push-down technique has been researched mainly for relational DSs, to the best of our knowledge, the applicability and efficiency of this technique on big data sources still faces mulitple open issues. Second, the optimization of DI processes with UDFs has not yet been researched enough to provide advanced and acceptable solutions. This problem becomes especially challenging for UDFs made available as black-boxes. For BBUDFs, a still open research question is how to reason about performance and internal logic of such UDFs. Third, the existing data lineage techniques (both from research and industry) do not handle cases when lineage links are broken by temporary objects.

In this context, the main **goal of the paper** is to point out to **research methods for managing DI processes** with the focus on: (1) efficient push-down methods on non-relational DSs - as the method of increasing performance of DI processes, (2) learning the performance characteristics of black-box UDFs and, if possible, their internal logic - as the method of optimizing the execution of DI processes with BBUDFs, (3) discovering and managing data lineage in DI processes in the presence of temporary objects and UDFs, which broke lineage - with the aim of providing additional functionality to data governance. In this paper we formulate the following **research hypotheses** (with their verification methods detailed in Section 4).

H1: we foresee that it will be possible to build a push-down optimizer that will be able to: (1) assess whether a given DI task profits from being pushed-down and (2) propose efficient (alternative) implementations of a given task pushed-down into a particular DS. As a consequence, we envisage that the push-down technique applied to non-relational data sources will allow to increase performance of DI processes (i.e., reduce their execution time). Based on the developed execution cost models and implementation of code snippets, it will be possible to efficiently push-down typical DI tasks [19] into the most popular non-relational DSs.

H2: we foresee that it will be possible to label BBUDFs with known performance classes. First, these classes will be built by ML techniques from performance characteristics (e.g., RAM usage, CPU usage) of known UDFs. Second, performance characteristics of a given BBUDF will be collected and used to assign the BBUDF to one of the already known performance classes. Since similar code snippets expose similar performance characteristics [67], at some extent, a given performance class might represent a similar internal logic of code snippets. This approach will allow to reason (with a given probability) about an expected BBUDF performance and (in some cases) about its internal logic. As a consequence, it might be further possible to decide whether a given BBUDF could be reordered (in the spirit of [21]) and/or parallelized (in the spirit of [68]).

H3: we foresee to be able to develop techniques for discovering broken lineage links between data and data objects (e.g., tables, views, materialized views, stored procedures and functions). The techniques should offer the discovery of the broken lineage links from a given data object onward as well as from a given data object backwards. Every discovered link will be accompanied by a probability of its connection with another object. To this end, we envisage using ML and statistical modeling.

4. Possible solutions

In order to verify hypotheses **H1**-**H3** (and hopefully to prove them) we foresee to start researching and developing the solutions outlined in this section.

4.1. Hypothesis H1

To prove **H1** it is necessary to conduct research on the most popular non-relational DSs i.e., key-value, column-family, text, and graph. To this end, an execution cost model for each typical tasks being pushed-down (e.g., filtering, value transformation, structure transformation, data anonymization) should be formulated for each type of a given data source. This would allow us to design implementation code skeletons for tasks pushed-down into these storage systems.

There are two fundamental research questions on the push-down technique. Q1: whether a given DI task pushed-down will cause an overall system performance improvement? Q2: given that the answer for Q1 is positive, how to efficiently implement a given task in a given

DS? In order to **answer Q1**, one could train a binary classifier on various types of data describing DSs, DI tasks, and performance characteristics, as discussed below.

Features of a DS should include among others: (1) a DS type, its producer and software version, (2) the support for indexes, if any, (3) the type of a query optimizer, if any, (4) the support for parallel processing, (5) current deployment architecture - centralized or distributed, (6) physical parameters of hardware (e.g., the number of CPUs, main memory size, disk type).

Features of a DI task executed in a DI engine should include among others: (1) task type (e.g., filtering, joining, aggregating, window function), (2) performance characteristics of the task (CPU time, elapsed processing time, RAM usage, I/O usage) for various data volumes and types - they will be collected by means of experiments.

Features of a pushed-down DI task should include among others: (1) task type (e.g., filtering, joining, aggregating, window function), (2) performance characteristics of the task (CPU time, elapsed processing time, RAM usage, I/O usage) for various data volumes and types and various set ups of a DS.

Performance characteristics of each DS should include CPU time, elapsed processing time, RAM usage, I/O usage, for various workload types of DI processes. These characteristics will be collected and stored in a repository for DI processes without and with pushed-down tasks.

In order to **answer Q2** we envisage the following approaches. The fist one could be based on a **pre-defined library of code templates**. For a given DI task, there will be a few code alternatives available for each DS. The selection of a given code template will be based on the decision of another classifier, learned on performance characteristics of these code templates in a given DS. The second approach could utilize a **genetic algorithm** to evolve a given code template from the library. As in the classical genetic approach to producing code snippets, each code template would be evaluated by means of a fitness score, based on how efficiently it performs a specific task. The third solution could apply **Large Technology Models** (LTMs) for the generation of code snippets, e.g., [69, 70] of pushed-down tasks. In this solution one could use the aforementioned features of data sources, DI tasks, and their performance characteristics to prompt an LTM.

4.2. Hypothesis H2

Proving **H2** is based on the assumption that knowing the performance class of a BBUDF one can get some insights (with a certain level of probability) on the kind of operations being executed and performance (for example w.r.t. a data volume) of the BBUDF, by analyzing other (known) UDFs belonging to the same class. Moreover, understanding how to parallelize the BBUDF can also be figured out in some cases by analyzing the parallelization techniques applied to the known UDFs in the same class. Being able to apply parallelization to a given BBUDF is a major step towards its performance improvement.

The solution could be based on **machine learning** (ML) techniques. To support the solution, the following components are foreseen: (1) the repository of performance characteristics of known (white-box) UDFs, which include CPU and RAM usage as well as I/O and elapsed processing time, all represented as time series (TSs), (2) classification algorithms for TSs, and (3) similarity measures for TSs. We assume that classes of performance characteristics of known UDFs have been built on TSs obtained from excessive experiments in parallel and centralized computing architectures. The white-box UDF characteristics are labeled to reflect their performance classes with parallelization and without. Based on these characteristics, various classifiers will be built to classify BBUDFs based on their performance characteristics.

When a new BBUDF is made available in the system, its performance characteristics are collected by means of experiments. Having collected the characteristics, the classification algorithms assign the BBUDF a label representing one of the existing performance classes. We anticipate two alternative scenarios of TSs classification, namely: (1) a few independent classifiers and (2) an ensemble of classifiers (a.k.a. Time Series Forests).

Our prior preliminary work on this problem [67] showed that popular classification algorithms, namely: kNN, *ROCKET* [71], and *HIVE-COTE* [72] produced promising results on TSs classification. This approach can be substantially extended by enlarging the set of classifiers with two that recently have gained high popularity in the research community, i.e., multi-layer perceptron [73] and CatBoost [74]. Moreover, the models can be build based on much larger experiments and for much broader types of BBUDFs, implementing operations like those listed in [19] but on non-relational data.

Notice that in order to classify TSs, a similarity measure is needed. In [67], we used the *dynamic time warping* (DTW) similarity measure [75]. TSs being compared may differ in amplitude, length, and phase. For this reason it is necessary to normalize them before comparing [76, 77]. We foresee extensive validation not only DTW but also other advanced similarity measures, like motifs [78], shapelets [79], locality-sensitive hashing [80], possibly combined with DTW, as inspired by [81, 82].

To conclude, our previous works on discovering the internals of BBUDFs was based on a *try and error approach* [83], but it turned out to be applicable only to a small class of simple DI tasks (implementing filtering, projection, filtering+projection), as in general the problem transforms to the Boolean Satisfiability Problem (belonging to the NP-complete class) [84]. For this reason, in our opinion the only sound **solution to reason about the internals of BBUDFs should be based on ML techniques**.

4.3. Hypothesis H3

Verifying **H3** could be based on the relational data model due to the fact that: (1) it contains a well defined and rich set of commands in the Data Definition Language and (2) it is one of the most popular data models, and (3) broken lineage has not been studied for this model. Since it is possible to discover only verisimilar broken lineage links, two alternative discovery approaches are foreseen. The first one is based on ML, whereas the second one - on statistical modeling.

The **ML approach** uses two different probabilistic classifiers, where classes represent data objects and links between them represent possible lineage links. Each link is labeled with a probability of one object being connected to another object. This approach has already been verified on a small pilot project [85] and for some simple scenarios real broken links were discovered. However, this approach still needs to be substantially extended to: (1) handle more complex broken lineage links, with longer chains of dependencies between objects, (2) improve the obtained prediction quality (e.g., measure F1), (3) be learned on substantially larger and versatile database schemas, and (4) provide advanced and clear visualizations of lineage graphs and discovered broken links.

The **statistical approach** assumes applying the *Hidden Markov Model* (HMM) that by learning the patterns in data transformations can automatically generate lineage graphs of data objects. In this application of the HMM, *states* represent different data sets and structures of data objects (e.g., tables, views, materialized views), *observations* represent transformations applied to data and data objects, *transition probability* represents the probability of moving from one hidden state to another, and *emission probability* represents the likelihood of observing a particular transition. Thus, the model will discover and represent the sequence of data and data object states and transformations that occurred in a given DI process from a DS, through a staging area to a destination system, which in our case will be a data warehouse.

The model can be learned and tested on historical lineage links acquired from code repositories (e.g., GitHub) and recognized database benchmarks (e.g., tpc.org). At this stage of our research, we envisage using the popular *Baum-Welch* algorithm to tune the parameters of the HMM and the *Viterbi* algorithm for discovering the most likely sequence of transitions from one state to another [86].

5. Conclusions

From the analysis of the state of the art on managing DI processes, we draw the following conclusions.

- Push-down has been researched mainly for relational DSs. To the best of our knowledge, the applicability and efficiency of this technique on big data sources reveals multiple open issues.
- The optimization of DI processes with UDFs has not yet been researched enough to provide advanced and acceptable solutions. This problem becomes especially challenging for UDFs made available as black-boxes. For BBUDFs, a still open research question is how to reason about performance and internal logic of such UDFs.
  The fact, that we have previously been cooperating with IBM Software Lab Kraków (Poland) on pilot projects on: (1) developing techniques for push-down on NoSQL DSs [87, 88] and (2) learning about BBUDFs [67], shows that solutions to these problems are of interest and value for the IT sector.
- From our cooperation with the financial sector in Poland, we received a feedback clearly indicating that solutions for discovering data lineage links and visualizing them for large data repositories (like data warehouses) is of significant importance. Legacy systems typically do not maintain data lineage and there is an evident need to build and visualize such links in these systems. For example, a small DW in one of the companies in the financial sector includes over 11000 stored procedures and functions, with over 2 million of codes and over 5000 of (materialized) views. Building and visualizing lineage even for such a small legacy DW is challenging. The problem is aggravated with the presence of temporary objects and UDFs that break data lineage. Unfortunately, the existing data lineage techniques (both from research and industry) do not handle cases when lineage links are broken by temporary objects.

The aforementioned conclusions motivate us to address these three research problems in this position paper. We consider them as important missing solutions in the data integration research.

References

[1] R. Wrembel, A. Abelló, I. Song, DOLAP data warehouse research over two decades: Trends and challenges, Information Systems 85 (2019).

[2] S. Siddiqi, R. Kern, M. Boehm, SAGA: A scalable framework for optimizing data cleaning pipelines for machine learning applications, SIGMOD 1 (2023).

[3] T. Timakum, S. Lee, H. Hu, I. Song, M. Song, DOLAP: A 25 year journey through research trends and performance (invited talk), in: Int. Workshop

on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP), volume 3653, CEUR-WS.org, 2024.

[4] J. Zhu, Y. Mao, L. Chen, C. Ge, Z. Wei, Y. Gao, Fusionquery: On-demand fusion queries over multi-source heterogeneous data, VLDB Endowment 17 (2024).

[5] A. Bouguettaya, B. Benatallah, A. Elmargamid, Interconnecting Heterogeneous Information Systems, Kluwer Academic Publishers, ISBN 0792382161, 1998.

[6] P. Brezany, A. M. Tjoa, H. Wanek, A. Wöhrer, Mediators in the architecture of grid information systems, in: Int. Conf. on Parallel Processing and Applied Mathematics (PPAM), volume 3019 of *LNCS*, Springer, 2003.

[7] S. A. Errami, H. Hajji, K. A. E. Kadi, H. Badir, Spatial big data architecture: From data warehouses and data lakes to the lakehouse, Journal of Parallel and Distributed Computing 176 (2023).

[8] R. Hai, C. Koutras, C. Quix, M. Jarke, Data lakes: A survey of functions and systems, IEEE Transactions on Knowledge and Data Engineering 35 (2023).

[9] A. A. Harby, F. H. Zulkernine, Data lakehouse: A survey and experimental study, Information Systems 127 (2025).

[10] Z. Dehghani, Data Mesh: Delivering Data-Driven Value at Scale, O'Reilly, ISBN 1492092398, 2022.

[11] M. Hameed, F. Naumann, Data preparation: A survey of commercial tools, SIGMOD Record 49 (2020).

[12] Gartner, Magic quadrant for data integration tools, 2022.

[13] IBM InfoSphere Information Server, https://www.ibm.com/information-server, accessed Dec 2024, ????

[14] S. M. F. Ali, R. Wrembel, From conceptual design to performance optimization of ETL workflows: current state of research and open problems, The VLDB Journal 26 (2017).

[15] A. Simitsis, S. Skiadopoulos, P. Vassiliadis, The history, present, and future of ETL technology (invited), in: Int. Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP), volume 3369 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023.

[16] 2020 state of data science, https://know.anaconda.com/rs/387-XNW-688/images/Anaconda-SODS-Report-2020-Final.pdf; accessed Dec 2024, 2020.

[17] M. Owaida, G. Alonso, L. Fogliarini, A. Hock-Koon, P.-E. Melet, Lowering the latency of data processing pipelines through fpga based hardware acceleration, VLDB Endowment 13 (2019).

[18] A. Lerner, R. Hussein, A. Ryser, S. Lee, P. Cudré-Mauroux, Networking and storage: The next computing elements in exascale systems?, IEEE Data Engineering Bulletin 43 (2020).

[19] Product documentation: Infosphere information server 11.3, https://www.ibm.com/docs/en/iis/11.3?topic=jobs-processing-data, accessed Feb, 2025, ????

[20] Informatica, White paper: How to Achieve Flexible, Cost-effective Scalability and Performance through Pushdown Processing, 2007.

[21] A. Simitsis, P. Vassiliadis, T. K. Sellis, State-space optimization of ETL workflows, IEEE TKDE 17 (2005).

[22] A. Simitsis, P. Vassiliadis, T. Sellis, Optimizing etl processes in data warehouses, in: ICDE, 2005.

[23] N. Kumar, P. S. Kumar, An efficient heuristic for logical optimization of ETL workflows, in: VLDB Workshop on Enabling Real-Time Business Intelligence, 2010.

[24] R. Halasipuram, P. M. Deshpande, S. Padmanabhan, Determining essential statistics for cost based optimization of an ETL workflow, in: EDBT, 2014.

[25] X. Yu, M. Youill, M. E. Woicik, A. Ghanem, M. Serafini, A. Aboulnaga, M. Stonebraker, Pushdowndb: Accelerating a DBMS using S3 computation, CoRR abs/2002.05837 (2020).

[26] B. Konieczny, What's new in Apache Spark 3.1 - predicate pushdown for JSON, CSV and Apache Avro, https://www.waitingforcode.com/apache-spark-sql/what-new-apache-spark-3.1-predicate-pushdown-json-csv-apache-avro/read, accessed Feb, 2025, 2021.

[27] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, S. Babu, Starfish: A self-tuning system for big data analytics, in: Biennial Conf. on Innovative Data Systems Research (CIDR), volume 11, 2011.

[28] X. Liu, C. Thomsen, T. B. Pedersen, Mapreduce-based dimensional ETL made easy, VLDB Endowment 5 (2012).

[29] X. Liu, C. Thomsen, T. B. Pedersen, ETLMR: A highly scalable dimensional ETL framework based on mapreduce, Trans. Large-Scale Data- and Knowledge-Centered Systems 8 (2013).

[30] X. Liu, N. Iftikhar, An ETL optimization framework using partitioning and parallelization, in: ACM SAC, 2015.

[31] S. M. F. Ali, J. Mey, M. Thiele, Parallelizing user–defined functions in the etl workflow using orchestration style sheets, Int. Journal of Applied Mathematics and Computer Science 29 (2019).

[32] S. M. F. Ali, R. Wrembel, Towards a cost model to optimize user-defined functions in an ETL workflow based on user-defined performance metrics, in: European Conf. on Advances in Databases and

Information Systems (ADBIS), LNCS 11695, 2019.
[33] F. Hueske, M. Peters, M. J. Sax, A. Rheinländer, R. Bergmann, A. Krettek, K. Tzoumas, Opening the black boxes in data flow optimization, VLDB Endowment 5 (2012).
[34] F. Hueske, M. Peters, A. Krettek, M. Ringwald, K. Tzoumas, V. Markl, J.-C. Freytag, Peeking into the optimization of data flow programs with mapreduce-style udfs, in: ICDE, 2013.
[35] P. Große, N. May, W. Lehner, A study of partitioning and parallel UDF execution with the SAP HANA database, in: Conf. on Scientific and Statistical Database Management (SSDBM), 2014.
[36] A. Rheinländer, A. Heise, F. Hueske, U. Leser, F. Naumann, Sofa: An extensible logical optimizer for udf-heavy data flows, Information Systems 52 (2015).
[37] P. Holanda, M. Raasveldt, M. Kersten, Don't keep my udfs hostage - exporting udfs for debugging purposes, in: Simpósio Brasileiro de Banco de Dados, 2017.
[38] Y. Huang, Z. Wang, C. Li, Udon: Efficient debugging of user-defined functions in big data systems with line-by-line control, SIGMOD 1 (2023).
[39] V. Simhadri, K. Ramachandra, A. Chaitanya, R. Guravannavar, S. Sudarshan, Decorrelation of user defined function invocations in queries, in: Int. Conf. on Data Engineering (ICDE), 2014.
[40] A. Crotty, A. Galakatos, K. Dursun, T. Kraska, C. Binnig, U. Çetintemel, S. Zdonik, An architecture for compiling udf-centric workflows, VLDB Endowment 8 (2015).
[41] K. Ramachandra, K. Park, Blackmagic: Automatic inlining of scalar udfs into SQL queries with froid, VLDB Endowment 12 (2019).
[42] M. E. Schüle, J. Huber, A. Kemper, T. Neumann, Freedom for the sql-lambda: Just-in-time-compiling user-injected functions in postgresql, in: Int. Conf. on Scientific and Statistical Database Management (SSDBM), ACM, 2020.
[43] E. Friedman, P. M. Pawlowski, J. Cieslewicz, Sql/mapreduce: A practical approach to self-describing, polymorphic, and parallelizable user-defined functions, VLDB Endowment 2 (2009).
[44] K. Ramachandra, K. Park, K. V. Emani, A. Halverson, C. A. Galindo-Legaria, C. Cunningham, Froid: Optimization of imperative programs in a relational database, VLDB Endowment 11 (2017).
[45] M. Sichert, T. Neumann, User-defined operators: Efficiently integrating custom algorithms into modern databases, VLDB Endowment 15 (2022).
[46] K. Chasialis, T. Palaiologou, Y. Foufoulas, A. Simitsis, Y. E. Ioannidis, Qfusor: A UDF optimizer plugin for SQL databases, in: Int. Conf. on Data Engineering (ICDE), IEEE, 2024.
[47] A. Rheinländer, M. Beckmann, A. Kunkel, A. Heise, T. Stoltmann, U. Leser, Versatile optimization of udf-heavy data flows with sofa, in: SIGMOD, ACM, 2014.
[48] C. Yan, Y. Lin, Y. He, Predicate pushdown for data science pipelines, SIGMOD 1 (2023).
[49] What is data lineage?, https://www.ibm.com/topics/data-lineage, accessed Dec 2024, ????
[50] Y. R. Wang, S. E. Madnick, A polygen model for heterogeneous database systems: The source tagging perspective, in: Int. Conf. on Very Large Data Bases (VLDB), 1990.
[51] L. Chiticariu, W.-C. Tan, G. Vijayvargiya, DBNotes: a post-it system for relational databases based on provenance, in: SIGMOD, 2005.
[52] J. N. Foster, T. J. Green, V. Tannen, Annotated XML: queries and provenance, in: ACM SIGMOD-SIGACT-SIGART symposium on Principles of Database Systems (PODS), 2008.
[53] P. Senellart, Provenance and Probabilities in Relational Databases, SIGMOD Record 46 (2018).
[54] D. Dosso, S. B. Davidson, G. Silvello, Data provenance for attributes: attribute lineage, in: USENIX Conf. on Theory and Practice of Provenance, TAPP, USENIX Association, 2020.
[55] Y. Cui, J. Widom, Practical lineage tracing in data warehouses, in: Int. Conf. on Data Engineering (ICDE), 2000.
[56] A. Woodruff, M. Stonebraker, Supporting fine-grained data lineage in a database visualization environment, in: Int. Conf. on Data Engineering (ICDE), 1997.
[57] M. Yamada, H. Kitagawa, T. Amagasa, A. Matono, Augmented lineage: traceability of data analysis including complex UDF processing, The VLDB Journal 32 (2023).
[58] P. Buneman, S. Khanna, W. C. Tan, Why and where: A characterization of data provenance, in: Int. Conf. on Database Theory (ICDT), volume 1973 of *LNCS*, 2001.
[59] O. Benjelloun, A. D. Sarma, A. Y. Halevy, M. Theobald, J. Widom, Databases with uncertainty and lineage, VLDB Journal 17 (2008).
[60] O. Benjelloun, A. D. Sarma, A. Y. Halevy, J. Widom, ULDBs: Databases with uncertainty and lineage, in: Int. Conf. on Very Large Data Bases (VLDB), ACM, 2006.
[61] D. Bhagwat, L. Chiticariu, W. C. Tan, G. Vijayvargiya, An annotation management system for relational databases, VLDB Journal 14 (2005).
[62] L. Chiticariu, W. C. Tan, G. Vijayvargiya, Dbnotes: a post-it system for relational databases based on provenance, in: Int. Conf. on Management of Data (SIGMOD), 2005.
[63] B. Glavic, G. Alonso, Perm: Processing provenance and data on the same data model through

query rewriting, in: Int. Conf. on Data Engineering (ICDE), 2009.
[64] F. Psallidas, E. Wu, Smoke: Fine-grained lineage at interactive speed, VLDB Endowment 11 (2018).
[65] Y. Amsterdamer, D. Deutch, V. Tannen, Provenance for aggregate queries, in: ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), 2011.
[66] T. J. Green, G. Karvounarakis, V. Tannen, Provenance semirings, in: ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), 2007.
[67] M. Bodziony, B. Ciesielski, A. Lehnhardt, R. Wrembel, On reasoning about black-box UDFs by classifying their performance characteristics, in: Harnessing Opportunities: Reshaping ISD in the post-COVID-19 and Generative AI Era (ISD), 2024.
[68] S. M. F. Ali, R. Wrembel, Framework to optimize data processing pipelines using performance metrics, in: Inf. Conf. on Big Data Analytics and Knowledge Discovery (DaWaK), LNCS 12393, 2020.
[69] F. Monti, F. Leotta, J. Mangler, M. Mecella, S. Rinderle-Ma, Nl2processops: Towards llm-guided code generation for process execution, in: Business Process Management Forum (BPM), volume 526 of *LNBIP*, Springer, 2024.
[70] F. Mu, L. Shi, S. Wang, Z. Yu, B. Zhang, C. Wang, S. Liu, Q. Wang, Clarifygpt: A framework for enhancing llm-based code generation via requirements clarification, ACM Software Engineering 1 (2024).
[71] A. Dempster, F. Petitjean, G. I. Webb, ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels, Data Mining and Knowledge Discovery 34 (2020).
[72] M. Middlehurst, J. Large, M. Flynn, J. Lines, A. Bostrom, A. J. Bagnall, HIVE-COTE 2.0: a new meta ensemble for time series classification, Machine Learning 110 (2021).
[73] M. Kulyabin, P. A. Constable, A. E. Zhdanov, I. O. Lee, D. A. Thompson, A. K. Maier, Attention to the electroretinogram: Gated multilayer perceptron for ASD classification, IEEE Access 12 (2024).
[74] L. Zhang, D. Jánosík, Enhanced short-term load forecasting with hybrid machine learning models: Catboost and xgboost approaches, Expert Systems with Applications 241 (2024).
[75] D. J. Berndt, J. Clifford, Using dynamic time warping to find patterns in time series, in: Workshop Knowledge Discovery in Databases, AAAI Press, 1994.
[76] S. Pumma, W. Feng, P. Phunchongharn, S. Chapeland, T. Achalakul, A runtime estimation framework for ALICE, Future Generation Computer Systems 72 (2017).
[77] Á. B. Hernández, M. S. Pérez, S. Gupta, V. Muntés-Mulero, Using machine learning to optimize parallelism in big data applications, Future Generation Computer Systems 86 (2018).
[78] A. Mueen, Time series motif discovery: dimensions and applications, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 4 (2014).
[79] J. Grabocka, N. Schilling, M. Wistuba, L. Schmidt-Thieme, Learning time-series shapelets, in: ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD), 2014.
[80] M. Datar, N. Immorlica, P. Indyk, V. S. Mirrokni, Locality-sensitive hashing scheme based on p-stable distributions, in: ACM Symposium on Computational Geometry, 2004.
[81] M. Ceccarello, J. Gamper, Fast and scalable mining of time series motifs with probabilistic guarantees, VLDB Endowment 15 (2022).
[82] A. Charane, M. Ceccarello, J. Gamper, Shapelets evaluation using silhouettes for time series classification, in: Int. Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP), volume 3653 of *CEUR Workshop Proceedings*, 2024.
[83] M. Bodziony, H. Krzyzanowski, L. Pieta, R. Wrembel, On discovering semantics of user-defined functions in data processing workflows, in: BiDEDE @ ACM SIGMOD/PODS Conference, ACM, 2021.
[84] S. Malik, L. Zhang, Boolean satisfiability from theoretical hardness to practical success, Communcations of the ACM 52 (2009).
[85] M. Grocholewski, T. Gruszczyński, Technologia lineage dla obiektów i kodu w bazach danych: koncepcja, implementacja, weryfikacja eksperymentalna (in polish); Lineage technique for objects and code in databases: concept, implementation, and experimental evaluation, 2024. Master thesis at Poznan University of Technology.
[86] J. P. Coelho, T. M. Pinho, J. Boaventura-Cunha, Hidden Markov Models: Theory and Implementation using MATLAB, Data-Centric Systems and Applications, CRC Press, 2021.
[87] M. Bodziony, S. Roszyk, R. Wrembel, On evaluating performance of balanced optimization of ETL processes for streaming data sources, in: Int. Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP), volume 2572, CEUR-WS.org, 2020.
[88] M. Bodziony, R. Morawski, R. Wrembel, Evaluating push-down on NoSQL data sources: experiments and analysis paper, in: BiDEDE @ ACM SIGMOD/PODS Conference, ACM, 2022.