Declarative expression and optimization of data-intensive flows

Georgia Kougka and Anastasios Gounaris

Department of Informatics
Aristotle University of Thessaloniki, Greece
{georkoug,gounaria}@csd.auth.gr

Abstract. Data-intensive analytic flows, such as populating a datawarehouse or analyzing a click stream at runtime, are very common in modern business intelligence scenarios. Current state-of-the-art data flow management techniques rely on the users to specify the flow structure without performing automated optimization of that structure. In this work, we introduce a declarative way to specify flows, which is based on annotated descriptions of the output schema of each flow activity. We show that our approach is adequate to capture both a wide-range of arbitrary data transformations, which cannot be supported by traditional relational operators, and the precedence constraints between the various stages in the flow. Moreover, we show that we can express the flows as annotated queries and thus apply precedence-aware query optimization algorithms. We propose an approach to optimizing linear conceptual data flows by producing a parallel execution plan and our evaluation results show that we can speedup the flow execution by up to an order of magnitude compared to existing techniques.

1 Introduction

Data-intensive analytic flows are typically encountered in business intelligence scenarios and are nowadays attracting renewed interest, since they go beyond traditional Extract - Transform - Load (ETL) flows [19, 16]. ETLs are a special form of data flows used to populate a data warehouse with up-to-date, clean and appropriately transformed source records. They can be considered as a directed acyclic graph (DAG), similar to scientific and business workflows, capturing the flow of data from the sources to the data warehouse [18]. Next generation business intelligence (BI) involves more complex flows that encompass data/text analytics, machine learning operations, and so on [16]; in this work we target such BI flows.

Our motivation is twofold. Firstly, modern data flows may be particularly complex to be described manually in a procedural manner (e.g., [16]). Secondly, the vast amount of data that such flows need to process under pressing time constraints calls for effective, automated optimizers, which should be capable of devising execution plans with minimum time cost. In this work, we target two correlated goals, namely declarative statement and efficient optimization.

Declarative statement of data flows implies that, instead of specifying the exact task ordering, flow designers may need to specify only higher-level aspects, such as the precedence constraints between flow stages, i.e., which task needs to precede other tasks. An example of an existing declarative approach is the Declare language that is based on linear temporal logic [12]. We follow a different approach that bears similarities with data integration mediation systems and allows the flow to be expressed in the form of annotated SQL-like queries.

Regardless of the exact declarative form a flow can be expressed, such declarative approaches are practical only under the condition that the system is capable of taking the responsibility for automatically devising a concrete execution plan in an efficient and dependable manner; this is exactly the role of query optimization in database systems, which also rely on declarative task specifications, and we envisage a similar role in data flow systems as well. Although traditional query optimization techniques cannot be applied in a straightforward manner, we propose an approach to optimizing linear conceptual data flows by producing a parallel execution plan, inspired by advanced query optimization techniques.

The contribution of this work is as follows. We demonstrate how we can express data flows in a declarative manner that is then amenable to optimization in a straight-forward manner. To this end, we use an annotated flavour of SQL, where flow steps are described by input and output virtual relations and annotations are inspired by the binding patterns in [5]. Our approach to declarative statement does not rely on the arguably limited expressiveness of relational algebra in order to describe arbitrary data manipulations, like those in ETLs, and is adequate to describe the precedence constraints between data flow tasks. In addition, we present optimization algorithms for logically linear flows that take into account the precedence constraints so that correctness is guaranteed. As shown in our evaluation, the approach that allows for parallel execution flows may lead to performance improvements up to an order of magnitude in the best case, and performance degradation up to 1.66 times in the worst case compared to the best current technique.

Structure: Sec. 2 presents our approach to declarative statement of data flows with a view to enabling automated optimization. The optimization algorithms for linear flows along with thorough evaluation of performance improvements are in Sec. 3. In Sec. 4 and 5 we discuss related work and conclusions, respectively.

2 Declarative statement of data flows

We map each flow activity[1] to a virtual relation described by a non-changing schema. More specifically, each activity is mapped to a virtual annotated relation $R(A, a, p)$, where (i) R is the task's unique name that also serves as its identifier; (ii) $A = (A_1, A_2, \cdots, A_n)$ is the list of input and output attributes, which are also identified by their names; (iii) a is a vector of size equal to the size of A, such that the i-th element of a is "b" (resp. "f") if the i-th element of A must be *bound* (resp. *free*); and (iv) p is a list of sets, where the j-th set includes the names of the *bound* variables of other virtual relations that must precede $R.A_j$.

[1] The terms flow tasks and activities are used interchangeably.

The notation of the a vector is aligned to the notation of *binding patterns* in [5], and allows us to distinguish between the attributes that need to belong to the input (the *bound* ones) and the new attributes that are produced in the output (*free* attributes). In other words, a binding pattern for a relation R means that the attributes of R annotated with b must be given as inputs when accessing the tuples of R, whereas the attributes annotated by f denote the new attributes derived by the task invocation. For example, the relation $Task1(A : (X, Y, Z),\ a : (bbf),\ p : (\{Task2.X\}\{NULL\}\{NULL\}))$ corresponds to an activity called *Task1*, which needs to be given the values of the X and Y attributes as input and returns a new attribute Z. Attribute X must first be processed by $Task2$. For brevity, this relation can also be written as $Task1(X^{b,Task2}, Y^b, Z^f)$ Additionally, we treat data sources as specific data-producing activities, where all attributes are annotated with f. Linear conceptual flows comprise a single data source.

The following statements hold: (a) The output data items of each flow task are regarded as tuples, the schema of which conforms to the virtual relations introduced above. (b) Data sources are treated as specific data-producing activities, where all attributes are annotated with f. (c) The flow tasks, even when they can be described by standard relational operators (e.g., when they simply filter data), they are always described as virtual relations. (d) The relations can be combined with standard relation operators, such as joins and unions; concrete examples are given in the sequel. (e) For each attribute X that is *bound* in relation R, there exists a relation R', which contains attribute X with a *free* annotation. (f) A task outputing a *free* attribute must precede the tasks that employ the same attributes as *bound* attributes in their schema. (g) Simply relying on b/f annotations is inadequate for capturing all the precedence constraints in ETL workflows, where there may exist a *bound* attribute that is manipulated by a filtering task and also appears in the *bound* grouping values of an aggregate function: in that case, the semantics of the flow may change if we swap the two activities, as also shown in [6]. For that reason, it is always necessary to define the p list of each activity. (h) Although most ETL transformation can be described by static schemas, there may be data flow activities, such as some forms of pivots/unpivots [3] that cannot be mapped to the virtual relations as defined above, because the schema of their output cannot be always defined *a-priori*. (i) Tasks need not correspond to ETL transformation solely; they can also encompass intermediate result storage.

Precedence constraints of a flow form a directed acyclic graph (DAG) G in which there is a node corresponding to each flow task and directed edges from one task to another define the presence of precedence constraint between them. A main goal of the annotations is to fully capture the precedence constraints among tasks. This goal is attained because the edges in the precedence graph can be derived from the p list of each activity and the (f) item above.

2.1 Flow Examples

Linear Flows Our first example is taken from [18] and is illustrated in Fig. 1. It is a linear flow that applies a set of filters, transformations, and aggregations to a

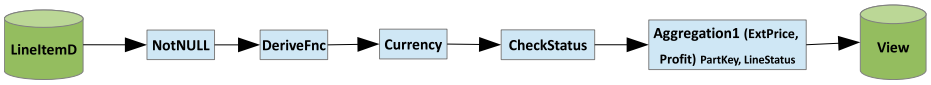

Fig. 1. A linear ETL flow.

single table from the TPC-H decision support benchmark. In particular, the flow consists of 5 activities: (i) *NotNull*, which checks the fields *PartKey, OrderKey* and *SuppKey* for NULL values. Any NULL values are replaced by appropriate special values (or the tuple is dropped). (ii) *DeriveFnc*, which calculates a value for the new field *Profit* that is derived from other fields and more specifically by subtracting the values of fields *Tax* and *Discount* from the value in *ExtPrice*. (iii) *Currency*, which alters the fields *ExtPrice, Tax, Discount* and *Profit* to a different currency. (iv) *CheckStat*, which is a filter that keeps only records whose return status is `false`. (v) *Aggregation1*, which calculates the sum of *ExtPrice* and *Profit* fields grouped by values in *PartKey* and *LineStatus*.

All activities can be mapped to virtual relations, and the whole ETL can be modelled as a Select-Project-Join (SPJ) query in order to provide online updates to the view in Fig. 1. It is important to note that the relevant attributes of the source relation, *LineItem*, are annotated as *free* attributes. Also, the $PartKey^{b,CheckStat}$ attribute in the aggregation activity contains a task annotation, which allows us to define that the aggregation must only be performed after the *CheckStat* activity in order to ensure semantic equivalence with the flow in Fig. 1. Finally, in *Aggregation1*, the attributes *PartKey* and *LineStatus* have the same values with *PartKeyGroup* and *LineStatusGroup*, respectively, but the latter are annotated as *free* attributes in order to facilitate manipulation statements that build on the grouped values.

```
Select PartKeyGroup, LineStatusGroup, UpdatedSumProfit,
    UpdatedSumExtPrice
From   LineItem(PartKey^f, OrderKey^f, SuppKey^f, Discount^f, Tax^f, ExtPrice^f, ···) ⋈
       NotNull(PartKey^b, OrderKey^b, SuppKey^b) ⋈
       DeriveFnc(PartKey^b, Profit^f) ⋈
       Currency(PartKey^b, ExtPrice^b, Discount^b, Profit^b, Tax^b) ⋈
       CheckStat(PartKey^b, ReturnStatus^b ⋈
       Aggregation1(PartKey^{b,CheckStat}, LineStatus^b, Profit^b, ExtPrice^b,
                    ReturnStatus^b, LineStatusGroup^f, PartKeyGroup^f,
                    UpdatedSumProfit^f, UpdatedSumExtPrice^f)
Where   LineItem.PartKey = NotNull.PartKey and
        LineItem.PartKey = DeriveFnc.PartKey and
        LineItem.PartKey = Currency.PartKey and
        LineItem.PartKey = CheckStat.PartKey and
        CheckStat.PartKey = Aggregation1.PartKey
```

In the above example there are several precedence constraints that can automatically derived from the annotated query: *LineItem* must precede all other activities, *DeriveFnc* must precede *Currency* and *Aggregation1*, whereas *CheckStat* must precede *Agregation1* as well. Although those constraints seem restrictive, they do not preclude other flow structures, e.g., *CheckStat* to be applied earlier and *Currency* to be applied at the very end to decrease the number of total currency transformations.

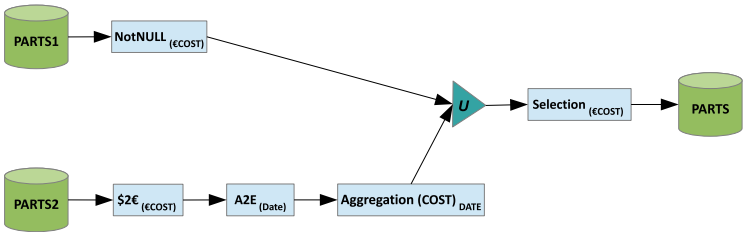

Fig. 2. A more complex ETL flow.

More complex flows Fig. 2 shows a more complex flow on top of two real data sources, also taken from [18]. The tasks employed are: (i) *NotNull*, which checks the field *Cost* for NULL values, so that such values are replaced or the tuple is dropped. (ii) *dollar2euro*, which changes the values in *Cost* from dollars to euros. (iii) *A2E*, which alters the format of the field *Date* from american to european. (iv) *Aggregation*, which calculates the sum of costs grouped by date, (v) *Selection*, which filters the (aggregated) cost field according to a user-defined threshold. Although we can describe this flow as a complex nested query, for clarity, we use two SPJ sub-queries. Note that, if the flow contains branches, these can be modeled as separate sub-queries in a similar manner. Also, although the selection task can easily be described by a simple select relational operator, we treat it as a separate relation.

Query I:
WITH Q ($PKEY$, $COST$, $DATE$) AS (
(Select *
From PARTS1($PKEY^f$, $COST^f$, $DATE^f$) $\bowtie$
NotNULL($PKEY^b$, $COST^b$)
Where PARTS1.PKEY = NN.PKEY)
UNION
(Select $PKEY$, $UpdatedAggCOST$, $DATEgroup$
From PARTS2($PKEY^f$,$COST^f$,$DATE^f$) $\bowtie$
dollar2euro($PKEY^b$,$COST^b$) $\bowtie$
A2E($PKEY^b$,$DATE^b$) $\bowtie$
Aggregation($PKEY^b$,$DATE^b$,$COST^b$,$DATEgroup^f$,$UpdatedAggCOST^f$)
Where PARTS2.PKEY = dollar2euro.PKEY and
PARTS2.PKEY = A2E.PKEY and
PARTS2.PKEY = Aggregation.PKEY)
)
Query II:
(Select *
From Q ($PKEY^f$, $COST^f$, $DATE^f$) $\bowtie$
Selection ($PKEY^b$, $COST^b$)
Where Q.PKEY = Selection.PKEY)

Real-world analytic flow The data flow, which is depicted in Fig. 3, shows a real-world, analytic flow that combines streaming free-form text data with structured, historical data to populate a dynamic report on a dashboard [16]. The report combines sales data for a product marketing campaign with sentiments about that product gleaned from tweets crawled from the Web and lists total sales and average sentiment for each day of the campaign. There is a single streaming source that outputs tweets on products and the flow accesses four other static sources through lookup operations.

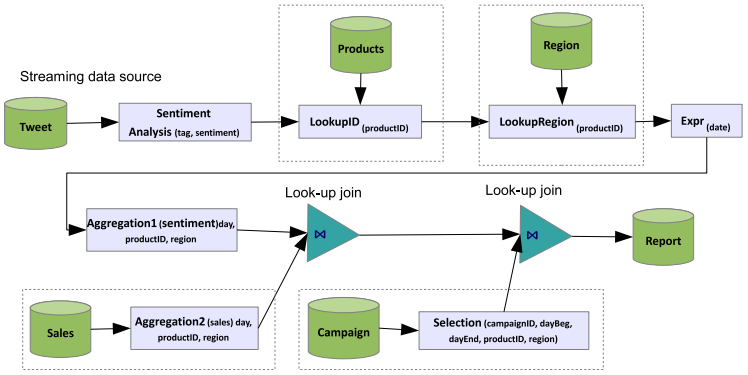

Fig. 3. A real-world analytic flow.

The exact flow is as follows. When a tweet arrives as a timestamped string attribute (*tag*), the first task is to compute a single sentiment value in the range [-5 +5] for the product mentioned in the tweet. Then, two lookup operations are performed: the former maps product references in the tweet and the later maps geographic information (latitude and longitude) in the tweet to a geographical region (*region* attribute in the figure). The *Expr* task converts the tweet timestamp to a date. Then, the sentiment values are averaged over each region, product, and date. On a parallel path, the sales data have been rolled up to produce total sales of each product for each region and day. The rollups for sales and sentiment are joined in a pipelined fashion and finally the specific campaign of interest is selected and used to filter the result based on the information of the campaign data store [16]. In this final stage, we consider that the *Sales* and *Campaign* non-streaming sources are hidden behind the *Aggregation2* and *Selection* look-up tasks, respectively. The annotated query that describes this flow is shown below.

```
Select *
From   Tweet(tag^f, timestamp^f) ⋈
       Sentiment_Anal(tag^b, sentiment^f) ⋈
       LookupID(tag^b, productID^f) ⋈
       LookupRegion(tag^b, region^f) ⋈
       Expr(tag^b, timestamp^b, date^f) ⋈
       Aggregation1(tag^b, sentiment^b, productID^b, region^b, date^b,
                    productIdGroup^f, dateGroup^f, regionGroup^f, AvgAggSentiment^f) ⋈
       Aggregation2(productID^f, region^f, date^f, totalAggSales^f) ⋈
       Selection(productID^f, campaignID^f, dayBeg^f, dayEnd^f, region^b)
Where  Tweet.tag = Sentiment_Anal.tag and
       Tweet.tag = LookupID.tag and
       Tweet.tag = LookupRegion.tag and
       Tweet.tag = Expr.tag and
       Tweet.tag = Aggregation1.tag and
       Aggregation1.productID = Aggregation2.productID and
       Aggregation1.region = Aggregation2.region and
       Aggregation1.date = Aggregation2.date and
       Aggregation1.productID = Selection.productID and
       Aggregation1.region = Selection.region
```

2.2 Are data flows queries?

The consensus up to now is that ETL and more generic data flows cannot be expressed as (multi-) queries, due to facts such as the presence of arbitrary manipulation functions that cannot be described by relational operators, and the presence of precedence constraints [4, 13]. We agree that data flows cannot be described as standard SQL queries just by regarding manipulation functions as black box user-defined functions (UDFs). Nevertheless, as shown above, we can express data flows in an *SQL-like* manner, where the distinctive features are that (i) data manipulation steps are described through virtual relations instead of relational operators or UDFs on top of real relations; and (ii) the attributes are annotated so that precedence constraints can be derived. Our methodology thus does not suffer from the limitations when mapping a flow to a complex query with as many relations as the original data sources, which loses the information about precedence constraints.

3 Optimization of linear flows

Having transformed the flow specification to an annotated query, we can treat the flows as multi-source precedence-aware queries and benefit from any existing optimization algorithms tailored to such settings. We treat flow tasks as black-box operators. Note that we do not have to use multi-way joins regardless of the numerous joins appearing in the SQL-like statements, as in [17]. In this section, we firstly define the cost model, we then propose four optimization algorithms for minimizing the total execution cost in time units, and finally, we investigate the performance benefits.

Our optimization algorithms require that each flow activity is described by the following metadata:

- *Cost* (c_i): We use $c_i = 1/r_i$ to compute response time effectively, where r_i is the maximum rate at which results of invocations can be obtained from the i-th task.
- *Selectivity* (sel_i): it denotes the average number of returned tuples per input tuple for the i-th service. For filtering operators selectivity is always below 1, for data sources and operators that just manipulate the input, it is exactly one, whereas, for operators that may produce more output records for each input record, the selectivity is above 1.
- *Input* (I_i): The size of the input of the i-th task in number of tuples per input data tuple. It depends on the product of the selectivities of the preceding tasks in the execution plan.

Our aim is to minimize the sum of the execution time of each task. As such, the optimal plan minimizes the following formula: $(I_i c_i + I_2 c_2 + ... + I_n c_n)$.

In the following, we present our optimization approaches. Due to lack of space, we present only the main rationale.

PGreedy: The rationale of the *PGreedy* optimization algorithm is to order the flow tasks in such a way that the amount of data that is received by expensive

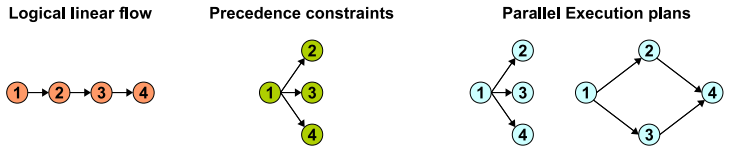

Fig. 4. A single linear conceptual data flow (left), along with its precedence constraints (middle) and two logically equivalent parallel execution plans (right).

tasks is reduced because of preceding filtering activities that prune the input dataset. Its main distinctive feature is that it allows for parallel execution plans, as shown in Fig. 4, where on the left part of the picture, a linear flow and its precedence constraints are depicted, while on the right two equivalent parallel execution plans of the same flow are presented (which both preserve all the precedence constraints). More specifically, depending on the selectivity values, the optimal execution plan may dispatch the output of an activity to multiple other activities in parallel, or place them in a sequence. To this end we adapt the algorithm in [17] with the difference that instead of considering the cost $I_i c_i$ in each step, we consider the $(1 - sel_i)(I_i c_i)$. The latter takes into account the selectivity of the next service to be appended in the execution plan and not only the selectivity of the preceding services. We refer the reader to [17] for the rest of the details. The complexity is $O(n^5)$ in the worst case.

Swap: The *Swap* algorithm compares the cost of the existing execution plan against the cost of the transformed plan, if we swap two adjacent tasks provided that the constraints are always satisfied. We perform this check for every pair of adjacent tasks. *Swap* is proposed in [15], where, to the best of our knowledge, the most advanced algorithm for optimizing the structure of data flows is proposed. The complexity of the *Swap* algorithm is $O(n^2)$.

Greedy: Greedy algorithm is based on a typical greedy approach by adding the activity with the maximum value of $(1 - sel_i)(I_i c_i)$, which meets the precedence constraints. The time complexity of *Greedy* algorithm is $O(n^2)$. It bears similarities with the *Chain* algorithm in [21]; latter appends the activity that minimizes $I_i c_i$. Similarly to *Swap* and contrary to *PGreedy*, it builds only linear execution plans.

Partition: The *Partition* optimization algorithm forms clusters with activities by taking into consideration their availability. Specifically, each cluster consists from activities that their prerequisites have been considered in previous clusters. After building the clusters, each cluster is optimized separately by checking each permutation of cluster tasks. Like *Greedy*, it was first proposed for data integration systems, and the details are given in [21]. *Partition* runs in $O(n!)$ time in the worst case, and is inapplicable if a local cluster contains more than a dozen of tasks.

3.1 Experiments

In our experiments, we compare the performance of the afore-mentioned algorithms. The data flows considered consist of $n = 5, 10, 25, 50, 100$ activities and we experiment with 6 combinations of 3 selectivity value ranges and 2 sets of

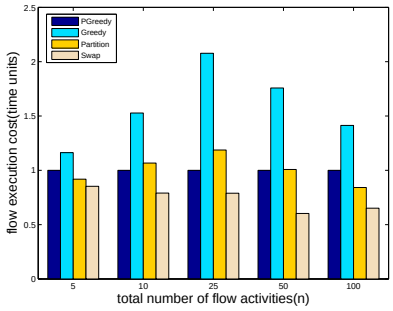

Fig. 5. Performance when $sel \in [0, 2]$, $cost \in [1, 10]$ and 25% prec. constraints.

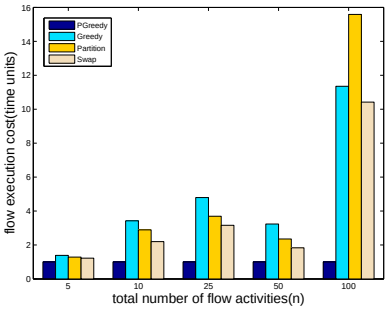

Fig. 6. Performance when $sel \in [0.5, 2.5]$, $cost \in [1, 10]$ and 25% prec. constraints.

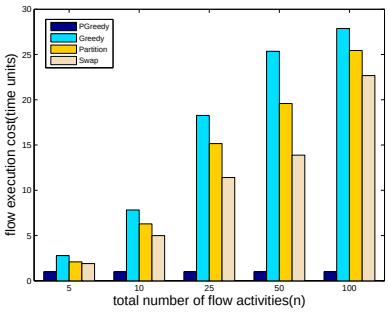

Fig. 7. Performance when $sel \in [1, 3]$, $cost \in [1, 10]$ and 25% prec. constraints.

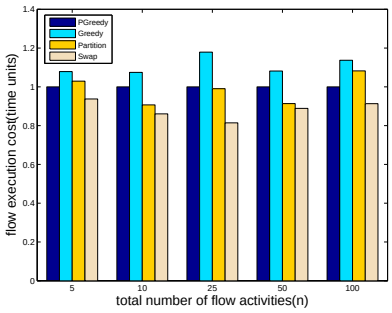

Fig. 8. Performance when $sel \in [0, 2]$, $cost \in [1, 10]$ and 50% prec. constraints.

constraint probabilities. The cost value range is the same for all the sets of experiments: $c_i \in [1, 10]$. The results correspond to the average of the data flow response time in 20 runs after removing the lowest and highest values to reduce the standard deviation. In each run, the exact selectivity, cost values and the constraints for each task are randomly generated.

In the first experiment, the selectivity values in each run are randomly generated so that $sel \in [0, 2]$ (thus only half of the tasks are selective) and $cost \in [1, 10]$ with 25% probability of having precedence constraints between two activities. The normalized results are shown in Fig. 5. A general observation in all our experiments is that *Swap* consistently outperforms *Greedy* and *Partition*. From Fig. 5, we can observe that *Swap* outperforms *PGreedy* as well. For $n = 50$, Swap is 1.66 times faster. However, as less activities are selective, *PGreedy* yields significantly lower cost than *Swap*. As shown in Figs. 6 and 7, those performance improvement may be up to 22 times (one order of magnitude).

In the following experiment, we increase the probability of having a precedence constraint between two activities. The more the constraints, the narrower the space for optimizations. The results are presented in Figs. 8-10, which follow the same pattern as above. In the worst case, *Swap* is 1.23 times faster than *PGreedy*, and, in the best case, *PGreedy* is 3.15 times faster. The general con-

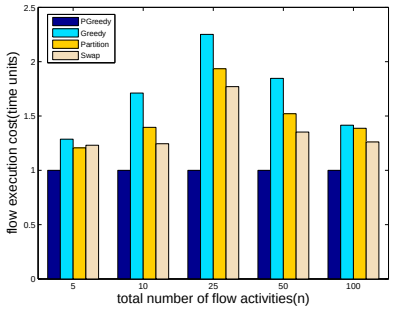

Fig. 9. Performance when $sel \in [0.5, 2.5]$, $cost \in [1, 10]$ and 50% prec. constraints.

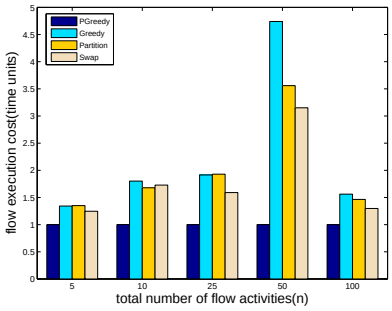

Fig. 10. Performance when $sel \in [1, 3]$, $cost \in [1, 10]$ and 50% prec. constraints.

clusions drawn is that *Greedy* and *Partition* are never the optimal choices, and *PGreedy* outperforms *Swap* if less than half of the tasks are selective.

Regarding the time needed for the optimizations, even when $n = 100$, the time for running *PGreedy* and *Partition* is approximately a couple of seconds using a machine with an Intel Core i5 660 CPU with 6GB of RAM. Thus it can be safely considered as negligible.

4 Related Work

Modern ETL and flow analysis tools, such as Pentaho's platform[2], do not support declarative statement of flows and automated optimizations of their structure. Declare is an example of a declarative flow language [12]; contrary to our proposal, it is based on linear temporal logic and can be used only through a graphical interface in the context of Yawl[3]. Declare can capture precedence constraints, and, as such, may stand to benefit from the optimizations proposed in this work, but does not perform any optimizations in its own right.

The potential of data management solutions to enhancing the state-of-the art in workflow management has been identified since mid 2000s. An example of strong advocates of the deeper integration and coupling of databases and workflow management systems has appeared in [14]. Earlier examples of developing data-centric techniques of manipulating workflows include the prototypes described in [7, 11, 9], which allow workflow tasks to be expressed on top of virtual data tables in a declarative manner but do not deal with optimization, although they can be deemed as enabling it. Other declarative approaches to specifying workflows, such as [1, 22], are not coupled with approaches to capturing precedence constraints and optimizing the flow structure either.

Data management techniques have been explored in the context of ETL workflows for data warehouses in several proposals, e.g., [4, 13, 15]. In [15], the authors consider ETL workflows as states and use transitions to generate new states in order to navigate through the state space. The main similarity with our work is the mapping of workflow activities to schemata, which, however,

[2] `http://www.pentaho.com/`
[3] `http://www.yawlfoundation.org/`

are not annotated and thus inadequate to describe precedence constraints on their own. Focusing on the physical implementation of ETL flows, the work in [18] exploits the logical-level description combined with appropriate cost models, and introduces sorters in the execution plans. In [16], a multi-objective optimizer that allows data flows spanning execution engines is discussed.

Another proposal for flow structure optimization has appeared in [20], which decreases the number of invocations to the underlying databases through task merging. In [10], a data oriented method for workflow optimization is proposed that is based on leveraging accesses to a shared database. In [6], the optimizations are based on the analysis of the properties of user-defined functions that implement the data processing logic. Several optimizations in workflows are also discussed in [2]. Our optimization approach shown in Section 3 is different from those proposals in that it is capable of performing arbitrary correct task reordering. In our previous work, we employ query optimization techniques to perform workflow structure reformations, such as reordering or introducing new services in scientific workflows [8].

5 Conclusions

As data flows become more complex and come with requirements to deliver results under pressing time constraints, there is an increasing need for more efficient management of such flows. In this work, we focused on data-intensive analytic flows that are typically encountered in business intelligence scenarios. To alleviate the burden to manually design complex flows, we introduced a declarative way to specify such flows at a conceptual level using annotated queries. A main benefit from this approach is that the flows become amenable to sophisticated optimization algorithms that can take over the responsibility for optimizing the structure of the data flow while taking into account any precedence constraints between flow activities. We discuss optimization of linear conceptual data flows, and our evaluation results show that we can speedup the flow execution by up to an order of magnitude if we consider parallel execution plans. Our future work includes the deeper investigation of optimization algorithms to non-linear conceptual flows and the coupling of optimization techniques that reorder tasks with resource scheduling and allocation in distributed settings.

Acknowledgements: This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

References

1. K. Bhattacharya, R. Hull, and J. Su. A data-centric design methodology for business processes. In *Handbook of Research on Business Process Modeling, chapter 23*, pages 503–531, 2009.
2. M. Böhm. *Cost-based optimization of integration flows.* PhD thesis, 2011.
3. Conor Cunningham, Goetz Graefe, and César A. Galindo-Legaria. Pivot and unpivot: Optimization and execution strategies in an rdbms. In *VLDB*, pages 998–1009, 2004.

4. U. Dayal, M. Castellanos, A. Simitsis, and K. Wilkinson. Data integration flows for business intelligence. In *Proc. of the 12th Int. Conf. on Extending Database Technology: Advances in Database Technology, EDBT*, pages 1–11. ACM, 2009.
5. D. Florescu, A. Levy, I. Manolescu, and D. Suciu. Query optimization in the presence of limited access patterns. In *Proc. of the 1999 ACM SIGMOD Int. Conf. on Management of data*, pages 311–322, 1999.
6. F. Hueske, M. Peters, M. Sax, A. Rheinländer, R. Bergmann, A. Krettek, and K. Tzoumas. Opening the black boxes in data flow optimization. *PVLDB*, 5(11):1256–1267, 2012.
7. Y.E. Ioannidis, M. Livny, S. Gupta, and N. Ponnekanti. Zoo: A desktop experiment management environment. In *Proc. of 22th Int. Conf. on Very Large Data Bases VLDB*, pages 274–285, 1996.
8. G. Kougka and A. Gounaris. On optimizing work ows using query processing techniques. In *SSDBM*, pages 601–606, 2012.
9. D.T. Liu and M.J. Franklin. The design of GridDB: A data-centric overlay for the scientific grid. In *VLDB*, pages 600–611, 2004.
10. R. Minglun, Z. Weidong, and Y. Shanlin. Data oriented analysis of workflow optimization. In *Proc. of the 3rd World Congress on Intelligent Control and Automation, 2000 - Volume 4*, pages 2564 – 2566. IEEE Computer Society, 2000.
11. S. Narayanan, U. V. Catalyrek, T. M. Kurc, X. Zhang, and J.H. Saltz. Applying database support for large scale data driven science in distributed environments. In *Proc. of the 4th Workshop on Grid Computing*, 2003.
12. Maja Pesic, Dragan Bosnacki, and Wil van der Aalst. Enacting declarative languages using LTL: Avoiding errors and improving performance. In *Model Checking Software*, pages 146–161. 2010.
13. T. K. Sellis and A. Simitsis. Etl workflows: From formal specification to optimization. In *ADBIS*, pages 1–11, 2007.
14. S. Shankar, A. Kini, D.J. DeWitt, and J. Naughton. Integrating databases and workflow systems. *SIGMOD Rec.*, 34:5–11, September 2005.
15. A. Simitsis, P. Vassiliadis, and T. K. Sellis. State-space optimization of etl workflows. *IEEE Trans. Knowl. Data Eng.*, 17(10):1404–1419, 2005.
16. A. Simitsis, K. Wilkinson, M. Castellanos, and U. Dayal. Optimizing analytic data flows for multiple execution engines. In *Proc. of the 2012 ACM SIGMOD Int. Conf. on Management of Data*, pages 829–840, 2012.
17. U. Srivastava, K. Munagala, J. Widom, and R. Motwani. Query optimization over web services. In *Proc. of the 32nd Int. Conference on Very large data bases VLDB*, pages 355–366, 2006.
18. V. Tziovara, P. Vassiliadis, and A. Simitsis. Deciding the physical implementation of ETL workflows. In *Proc. of the ACM 10th Int. Workshop on Data warehousing and OLAP DOLAP*, pages 49–56, 2007.
19. P. Vassiliadis and A. Simitsis. Near real time ETL. In *New Trends in Data Warehousing and Data Analysis*, pages 1–31. 2009.
20. Marko Vrhovnik, Holger Schwarz, Oliver Suhre, Bernhard Mitschang, Volker Markl, Albert Maier, and Tobias Kraft. An approach to optimize data processing in business processes. In *VLDB*, pages 615–626, 2007.
21. Ramana Yerneni, Chen Li, Jeffrey D. Ullman, and Hector Garcia-Molina. Optimizing large join queries in mediation systems. In *ICDT*, pages 348–364, 1999.
22. Y. Zhao, J. Dobson, I. Foster, L. Moreau, and M. Wilde. A notation and system for expressing and executing cleanly typed workflows on messy scientific data. *SIGMOD Rec.*, 34:37–43, 2005.