Multi-workflow optimization in PAW

Maxim Filatov
University of Geneva
maxim.filatov@unige.ch

Verena Kantere
University of Geneva
verena.kantere@unige.ch

ABSTRACT

As business decisions and strategies become more and more automated, real-time, and data-driven, enterprises need to create, manage and execute end-to-end analytics workflows that process increasing data volumes, from new heterogeneous data sources, on specialized processing engines. Designing and optimizing such workflows is a challenging task since they span a variety of systems and tools. To address these needs, we present the Platform for Analytics Workflows (PAW). PAW enables workflow design, execution, analysis and optimization with respect to time efficiency, over multiple execution engines and storage repositories. In this paper, we focus on the demonstration of the functionality of PAW related to multi-workflow optimization. We demonstrate the functionality of PAW for users with various expertise and its capabilities with respect to workflow analysis and optimization. We employ several scenarios of running workloads of workflows with and without PAW's optimization on real use cases and data from the telecommunication domain and web analytics, but also on synthetic use cases and data.

1. INTRODUCTION

The analysis of Big Data is a core and critical task in multifarious domains of science and industry. Such analysis needs to be performed on a range of data stores, both traditional and modern, on data sources that are heterogeneous in their schemas and formats, and on a diversity of query engines. Moreover, such analysis is also intensive and systematic. This means that many users access the same data at the same time with different or similar target results, and, such results are the output of an analytics process. Thus, a system that enables such analytics processes on Big Data needs to be able to manage several workflows and execute them in an optimal manner. Workflow execution can be extremely resource- and time-consuming. Therefore, the optimization of the execution of a single workflow but also of the joint execution of several workflows is very important for the efficiency of such a system.

Commercial Extract-Transform-Load (ETL) tools (e.g. [3], [2]) provide little support for automatic optimization. They provide hooks for the ETL designer to specify for example which flows may run in parallel or where to partition flows for pipeline parallelism. Some ETL engines such as PowerCenter [3] support PushDown optimization, which pushes operators expressed in SQL from the ETL flow down to the source or target database engine. The rest of the transformations are executed in the data integration server. The challenge of optimizing the entire workflow remains unsolved.

Towards this direction, HFMS [4] performs optimization and execution across multiple engines. Work related to HFMS [5] focuses on optimizing flows for several objectives: performance, fault-tolerance and freshness over multiple execution engines. HFMS uses many optimization strategies, such as parallelization, recovery points, function shipping, data shipping, decomposition, etc. However, HFMS does not focus on managing or optimizing in a joint manner multiple workflows.

We demonstrate a novel technique for multi-workflow optimization that is implemented as part of our system called PAW (Platform for Analytics Workflows), a platform for the design, analysis and execution of analytics workflows. To the best of our knowledge, there is no previous work on multi-workflow optimization. The first version of PAW is presented in [1]. A workflow created in PAW is prepared for execution in three steps: First, the tasks are analyzed and the workflow is augmented with associative tasks; the new version of the workflow, which we call the *analyzed* workflow, represents not only the logic flow of the analytics process but also its execution semantics. Second, workflows are manipulated by swapping, composing/decomposing and factorizing/distributing transitions, in order to achieve workflows that have equivalent outputs with their original state, but have a form that can result in optimized execution. Third, PAW schedules the execution of a set of workflows following the novel technique of multi-workflow optimization, on which we focus in this demonstration. This technique is based on the joint execution of the common parts of two or more workflows. PAW can be employed on top of any system that executes analytics processes on big data sources. The platform mediates between users and a set of available data management technologies, such as relational DBMSs, key-value stores and column stores.

2. OVERVIEW OF PAW

PAW is a part of a larger system, called Adaptable Scalable Analytics Platform (ASAP) [6], but it can also stand as an independent tool for workflow management and optimization. Other ASAP components include execution, monitoring, visualization of results, online adaptation, etc. PAW presents a unified interface for users to create, modify, analyze, optimize and execute analytics workflows over a diverse collection

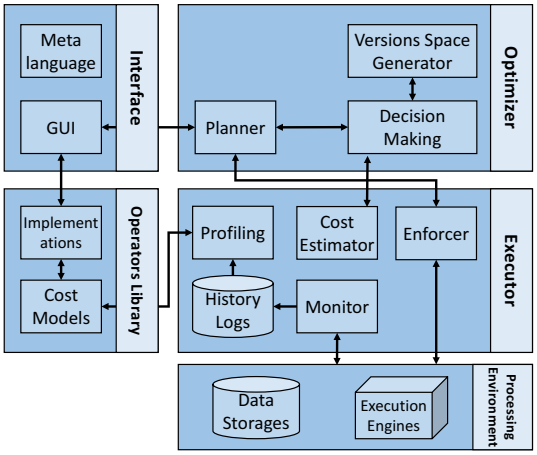

Figure 1: The architecture of PAW

of data stores and processing engines. Figure 1 depicts the architecture of PAW, as well as its interaction with the rest of ASAP. The components of PAW communicate using the internal workflow representation and are:

Operators library. This library contains operators, and their corresponding *implementations* with *cost functions*. The operators are classified as, either logical operators, which perform the core analytics jobs over the data, or the associative operators, which serve as 'glue' between different engines and perform move and transformation operations.

Interface. The *GUI* allows users to interactively create and/or modify a workflow, and add new operators to the *Library*. The user designs a workflow graph in the interactive tool and describes data and operators in the *Tree-metadata language*, which captures structural information, operator properties, and so on.

Optimizer. The orchestration of the optimization process is performed by the *Planner*. It takes as an input a workflow from the *Interface* and sends it to the *Decision Making* module, that returns back an optimized version of a workflow. All possible versions are produced in the *Versions Space Generator* and their costs are estimated by the *Cost Estimator*. The *Decision Making* module chooses the version with the minimal cost as an optimal one.

Executor. The executor performs several tasks. The *Enforcer* schedules workflows for execution, generates executable code and dispatches workflow fragments to execution engines. The *Monitor* observes the system state, tracks the progress of executing workflows and stores *History Logs* of runs. These logs are used to construct more precise *cost functions* of operators through the *Profiling* module. This module in PAW is external, it is also developed as a part of ASAP project, and called IRES [10].

PAW implements a novel workflow model [7, 8]. A workflow W is a directed, acyclic graph (DAG) $G = (V, E)$. The vertices V represent data processing tasks and the edges E represent the flow of data. Each task is a set of *inputs*, *outputs* and an *operator*. Data and operators need to be accompanied by a set of metadata, i.e., properties that describe them. Such properties include input data types and parameters of operators, the location of data objects or operator invocation scripts, data schemas, implementation details, engines etc.

3. MULTI-WORKFLOW OPTIMIZATION

Our technique of multi-workflow optimization (MWO) is based on the joint execution of the common parts of workflows. Specifically, a set of workflows is combined to one joint workflow, so that one or more common subgraphs in these workflows, appear only once in the joint workflow and, therefore, are executed only once. The technique consists of four steps: (1) for each workflow generate all possible equivalent workflow versions and prune them using heuristics; (2) detect common tasks and find the common parts in workflow versions; (3) estimate the processing cost of joint executions; (4) choose workflow versions and common parts in them for the joint execution.

3.1 Generating workflow versions

Two workflow versions are equivalent if they produce the same output, given the same input. We generate all possible versions by applying the following transitions:

Swap. The *swap* transition applies to a pair of vertices, v_1 and v_2, which occur in adjacent positions in a workflow graph G, and produces a new graph G' in which the positions of v_1 and v_2 have been interchanged. The goal of *swap* is to change the execution order of tasks.

Compose. The *compose* transition takes as input two vertices and produces one new vertex that includes the tasks of both initial vertices. The goal of *compose* is to allow for a united optimisation of the tasks included in the two vertices, e.g. joint micro-optimization on an execution engine.

Decompose. The *decompose* transition takes as input one vertex and produces two new vertices that, together, include all the tasks of the initial vertex. The new vertices may or may not be connected. The goal of *decompose* is to lead to separate optimisation of subgroups of the tasks.

Factorize. The *factorize* transition replaces multiple identical vertices that all feed (or are fed by) one *branching* vertex and take as input different datasets, with one such vertex that is performed on the output (input) data of the *branching* vertex. The optimization derives from the fact that the operation of the replaced vertices is performed only once instead of several times, and, moreover, on a reduced in size aggregated dataset.

Distribute. The *distribute* transition replaces one vertex with multiple identical ones, which are distributed on the input (or output) paths of a preceding (or succeeding) *branching* vertex. The optimization opportunity is created either by the parallelization of the execution of the identical vertices, their distribution over the input dataset, or even by the reduction of size of the aggregated input data due to their being pushed toward the root of the workflow.

If the version space is big, exploration methods more efficient than exhaustive search are required. We improve search performance by pruning the space with several heuristics based on the following categorization of operators:

- **Blocking operators** require knowledge of the whole dataset.
- **Non-blocking operators** process each tuple separately.
- **Restrictive operators** output a smaller data volume than the incoming data volume.

R1-2 are a list of rules, following which the process of generating the search space speeds up. Heuristics H1-2 prunes the search space.

- **R**1: Find branching operators and check if they are connected with operators that are identical instances of a logical operator. Try to *factorize* this set of operators.
- **R**2: Find (linear) paths and try to *swap* the operators in

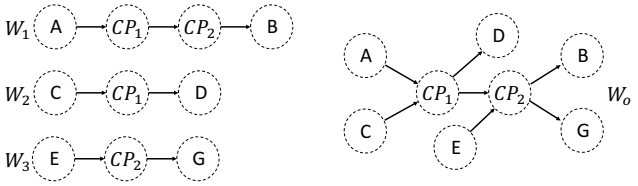

Figure 2: Example of multi-optimization of three workflows

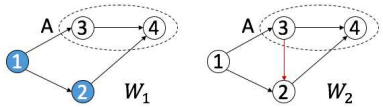

Figure 3: Independently executable and not independently executable subgraphs

each of such paths.

- **H**1: Move restrictive operators to the root of the workflow, e.g. change *extract* → *function* → *filter* to *extract* → *filter* → *function*, if possible.
- **H**2: Group non-blocking operators together and separately from blocking operators, e.g., change *filter* → *sort* → *function* → *group* to *filter* → *function* → *sort* → *group*.

3.2 Creating the joint workflow

A set of workflows $\mathcal{W} = \{W_1, \ldots, W_m\}$ may be combined in a joint workflow denoted as $W_o = W_1 \circ \cdots \circ W_m$. We find common parts in the workflows and use them as joint subgraphs connected with the rest of the workflow graphs. Figure 2 depicts three workflows W_1, W_2, W_3 and a joint workflow of them, W_o. CP_1 and CP_2 represent common parts of W_1, W_2 and W_1, W_3, respectively, and $A..G$ are the remaining parts of workflows.

3.2.1 Finding common parts

A common part consists of common tasks. Two common tasks consist of the same operators, inputs and outputs. We detect common tasks by comparing properties of metadata of tasks, such as input and output data schemas, parameters of operators etc.

After detecting common tasks, we look for subgraphs consisting only of common tasks and compare their structures. If such subgraphs are identical, then they constitute a *common part*. Formally, the latter is defined as follows:

Definition 1. A *common part* $CP(W_1, \ldots, W_m)$ of a set of workflows $\{W_1, \ldots, W_m\}$ is a subgraph S, so that S is part of every one of the workflows, i.e. $S \in W_1 \wedge \cdots \wedge S \in W_m$, and operators of corresponding vertices in a subgraph S of every workflow are identical.

3.2.2 Evaluation of a common part

After finding a common part, we determine if it can be used for the creation of the joint workflow. We do this based on the concepts of *execution state* and *independently executable subgraph*.

An *execution state* ES of a workflow W is a state for which some of the vertices are assumed to have been executed and no vertices are executing. An *independently executable subgraph* $S \in W$ with respect to some execution state ES_W, is a subgraph that can be executed without executing any vertex in $W \setminus (ES_W \cup S)$.

Figure 3 depicts two workflows W_1 and W_2. In W_1, subgraph A is independently executable with respect to the execution state, the executed vertices of which are colored in blue. In W_2, subgraph A is not independently executable with respect to any execution state, because vertex 4 cannot be executed before vertex 2, and vertex 2 cannot be executed before vertex 3, so vertex 2 has to be executed between vertices 3 and 4.

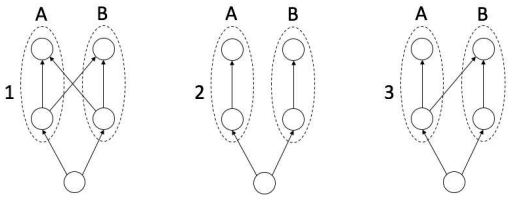

Figure 4: Mutual arrangement of subgraphs A and B

The creation of a joint workflow W_o of a set of workflows $\mathcal{W} = \{W_1, \ldots, W_m\}$ that have one common part CP, is possible if CP is independently executable for some execution state for every $W \in \mathcal{W}$.

3.2.3 Evaluation of a set of common parts

A set of workflows to be composed may contain not one, but several common parts. There can be cases for which not all of the common parts can be used for the creation of the joint workflow. To evaluate if a set of common parts $\mathcal{CP}$ can be used in combination for the creation of the joint workflow, we check the *mutual arrangement* of common parts in this set in pairs $CP_i, CP_j \in \mathcal{CP}$.

A *vertex v is reachable* from another vertex u if there is a directed path that starts from u and ends at v. A *subgraph S depends* on vertex v if there exists a vertex u in the subgraph and u reachable from v. The possible *mutual arrangement* of the subgraphs corresponding to two common parts CP_i and CP_j is one of the following (Figure 4):

1. Independent, if there does not exist a pair of vertices $\{v_i, v_j\}$, $v_i \in CP_i, v_j \in CP_j$ for which CP_i depends on v_j or CP_j depends on v_i.
2. CP_i depends on CP_j, if there is a vertex $v \in CP_j$ and CP_i depends on v, but there is not a vertex in CP_i so that CP_j depends on it.
3. CP_i and CP_j are cross-dependent if there are vertices $v_i \in CP_i$, $v_j \in CP_j$ and CP_j depends on v_i and CP_i depends on v_j.

Depending on their mutual arrangement in the set of workflows, a pair of common parts can be selected for the construction of the joint workflow or not: If the common parts are mutually arranged as (1) in all workflows, both can be selected; if they are mutually arranged as in (3), even in one workflow, they cannot be both selected. If they are mutually arranged as in (2) in some of the workflows, they can be both selected if they have the same dependency in all these workflows. Hence, in some cases, we are forced to select only some of the common parts. We do this based on the estimation of processing cost of different choices for the construction of the joint workflow.

3.3 Estimation of processing costs

We estimate the performance and cost of operators by actually running the operator in representative configuration combinations. Using these measurements, surrogate estimator models are trained that can be used to approximate operators performance for non-tested configurations. The processing cost of a workflow W, C_W, is the sum of the cost of the its tasks: $C_W = \sum_{i=1}^{n} C_{T_i}$.

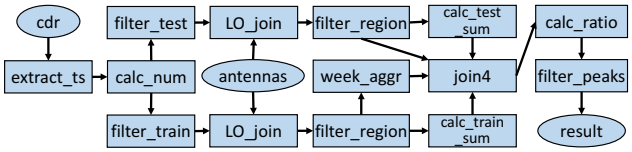

Figure 5: 'Peak Detection' workflow

Let us consider a pair of workflows $\{W_1, W_2\}$ with a common part CP and execution states ES_1 and ES_2, respectively. The cost of the joint workflow $W_o = W_1 \circ W_2$ is the sum of the cost of execution states $C(ES_1)$ and $C(ES_2)$, the cost of the common part $C(CP)$, the costs of the rest of workflows $C(W_1 \setminus \{CP, ES_1\})$, $C(W_2 \setminus \{CP, ES_2\})$, and a synchronization cost $C(sync)$, which captures the cost for creating the joint workflow:

$$C(W_1 \circ W_2) = C(ES_1) + C(ES_2) + C(CP) + C(sync) + \\ + C(W_1 \setminus \{CP, ES_1\}) + C(W_2 \setminus \{CP, ES_2\}) = \\ = C(W_1) + C(W_2) - C(CP) + C(sync)$$

The processing cost of workflows $\mathcal{W} = \{W_1, \ldots, W_m\}$ with common parts $\{CP_1, \ldots, CP_n\}$ is:

$$C(W_1 \circ \overset{m}{\cdots} \circ W_m) = \\ = \sum_{i=1}^{m} C(W_i) - \sum_{i=1}^{n}((n_i - 1)C(CP_i) - C(sync_i))$$

where n_i is the number of occurrences of common part CP_i in $\mathcal{W}$. After estimating the processing costs of all workflow versions and common parts, exhaustive search chooses common parts and workflows with the lowest cost.

3.4 Online Multi-Workflow Optimization

MWO is applicable, when the user launches multiple workflows simultaneously. Usually, a frequent case is when PAW receives new workflows one by one or in batches, while some workflows are currently executing. To cover this case PAW offers an Online Multi-Workflow Optimization (OMWO). It re-optimizes currently running workflows on each addition of a new workflow to our platform. As soon as a new workflow is inserted to PAW the optimizer gets the current states of execution of workflows, i.e. which vertices have been executed, are executing and have not yet started execution. Next, it applies MWO to a set of workflows, that consist of the new workflow and not-executed parts of workflows that are currently executing. Their intermediate results are used as inputs in these partial workflows.

4. DEMONSTRATION

In the following, we describe the proposed demonstration.

System setup. PAW is demonstrated on a cluster, with the following configuration: The cluster consists of 4 server-grade physical nodes. Each one of those is equipped with a 3rd generation i5 CPU (@ 2.90 GHz) and 16GB of physical memory and an array of two HDDs on RAID-0. The operating system is Debian 6 (squeeze) Linux. For the time being, three software platforms are running: Hadoop (CDH 4.6.0), Spark (1.4.1) and Weka (3.6.13).

Workloads. The demonstration uses synthetic and real workflows on real data. The synthetic workflows are constructed based on ETL benchmarking [9]. Real workflows and data come from the two use cases of ASAP [6] and belong to the domains of telecommunications and web analytics. Figure 5 displays one of the telecommunication workflows. The telecommunication use case involves processing anonymised Call Detail Records (CDR) data collected in Rome for 2015 year and stored in HDFS. All workflows' operators have implementations in Spark and Postgres. The web analytics use case involves anonymization of web content (WARC files) stored in ElasticSearch. The workflows are implemented in Spark and run over varying data set sizes ranging from 1 million to 4 billion rows. There are two types of workflows: one models entity recognition/disambiguation and k-means, and another models continuous processing of incoming data, e.g., subscription/notification at scale.

Demonstration scenarios. The demonstration focuses on the multi-workflow optimization functionality of PAW. It includes three types of scenarios that aim to show a distinct view of the benefit of our novel technique and create discussion on the potential of multi-workflow optimization. The demonstration is interactive with the audience. The participants are invited to experience all functionalities of PAW, create workflows from scratch or change existing ones, watch the automated management of the workflow as well as review the internals of the platform, e.g. internal workflow representation. They are also enabled to play with the management of multiple workflows, by selecting workflows for optimization and execution, pausing and resuming execution, selecting common parts for optimization, etc.

Scenarios A. Their goal is the comparison of single and multi-workflow optimization. We show exemplary cases of small sets of workflows, in which the versions selected by single-workflow optimization differ or identify with the versions selected by multi-workflow optimization.

Scenarios B. Their goal is to show the overall performance of multi-workflow optimization for a variety of workflow workloads. The workloads include workflows with a variety of tasks, short-running and long-running, in a variety of combinations, with an emphasis on long chain paths or numerous parallel paths. Also, the scenarios show how the existence of common parts affects optimization, by varying their number and their size.

Scenarios C. Their goal is to show the continuous arriving, optimization and execution of workflows. We create time series of workflows from scenarios B. We emphasize in the effect of ranging the size of the window on the arrival timeline, within which workflows are optimized by online multi-workflow optimization.

5. REFERENCES

[1] M. Filatov and V. Kantere. PAW: A Platform for Analytics Workflows. In *EDBT*, 2016.
[2] Oracle warehouse builder 10g. http://www.oracle.com/technology/products/warehouse/.
[3] Informatica 'powercenter'. http://www.informatica.com/products/powercenter/.
[4] A. Simitsis, K. Wilkinson, M. Castellanos, and U. Dayal. Optimizing analytic data flows for multiple execution engines. In *ACM SIGMOD*, 2012.
[5] A. Simitsis, K. Wilkinson, U. Dayal, and M. Hsu. HFMS: Managing the lifecycle and complexity of hybrid analytic data flows. In *ICDE*, 2013.
[6] Asap. http://www.asap-fp7.eu/.
[7] V. Kantere and M. Filatov. A framework for big data analytics. In *C3S2E*, 2015.
[8] V. Kantere and M. Filatov. Modelling processes of big data analytics. In *WISE*, 2015.
[9] A. Simitsis, P. Vassiliadis, U. Dayal and V. Tziovara. Benchmarking ETL workflows. In TPCTC, 2009.
[10] K. Doka, N. Papailiou, D. Tsoumakos and N. Koziris: IReS: Intelligent, Multi-Engine Resource Scheduler for Big Data Analytics Workflows. In SIGMOD, 2015.