Demonstration

PAW: A Platform for Analytics Workflows

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

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

ABSTRACT

Big Data analytics in science and industry are performed on a range of heterogeneous data stores, both traditional and modern, and on a diversity of query engines. Workflows are difficult to design and implement since they span a variety of systems. To reduce development time and processing costs, automation is needed. We present PAW, a platform to manage analytics workflows. PAW enables workflow design, execution, analysis and optimization with respect to time efficiency, over multiple execution engines, namely a DBMS, a MapReduce engine, and an orchestration engine. This configuration is emerging as a common paradigm used to combine analysis of unstructured data with analysis of structured data (e.g., NoSQL plus SQL). The demonstration of PAW focuses on the usability of the platform by users with various expertise, the automation of the analysis and optimization of execution, as well as the effect of optimization on workflow execution. The demonstration scenarios are based on synthetic and real workflows on real data.

1. INTRODUCTION

Enterprises today employ a variety of data repositories and processing engines to meet their needs for analytics. Analytics workflows are becoming longer and more complex. Currently, analytics workflows are designed and implemented manually. This is time-consuming and labor-intensive. To address this, we demonstrate a platform to automate part of this process.

Workflow management is not a new topic [17]. However workflow optimisation is a relatively new field of research, but there are already some promising results.

Commercial Extract-Transform-Load (ETL) tools (e.g. [5], [10]) 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 [5] support PushDown optimization, which pushes operators that can be expressed in SQL from the ETL flow

©2016, Copyright is with the authors. Published in Proc. 19th International Conference on Extending Database Technology (EDBT), March 15-18, 2016 - Bordeaux, France: ISBN 978-3-89318-070-7, on OpenProceedings.org. Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0

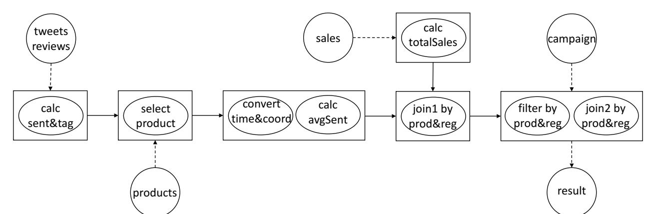

Figure 1: Workflow for a product marketing campaign

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.

Towards this direction, HFMS [13] performs optimization and execution across multiple engines. Work related to HFMS [14] 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. Complementary to the above, our work focuses on the automation of the total process of workflow manipulation, from the creation till the execution of a workflow. Furthermore, our work focuses on the challenge of enabling users with various levels of data management expertise to create a workflow for the same application logic.

In this paper, we demonstrate our work through PAW, a platform for the design, analysis and execution of analytics workflows. PAW implements a novel workflow language [7, 8] that allows the design of a workflow that spans multiple engines and data stores by either giving specific details on execution semantics of tasks and data stores or leaving the platform to determine the execution semantics and data stores, through an automated workflow analysis phase. Then, the workflow goes through an automated optimization phase, before being sent for execution. PAW is part of the ASAP project [1], which develops scalable solutions for complex analytical tasks over multi-engine environments.

In the following, Sections 2 and 3 give an overview of the workflow model and the platform architecture, respectively. Section 4 summarizes the functionalities of the platform and Section 5 describes the proposed demonstration.

2. WORKFLOW MODEL

PAW implements a novel workflow model [7, 8]. The workflows are directed, acyclic graphs (DAGs). The vertices represent data processing and the edges represent the flow of data. Data processing is computation or modification

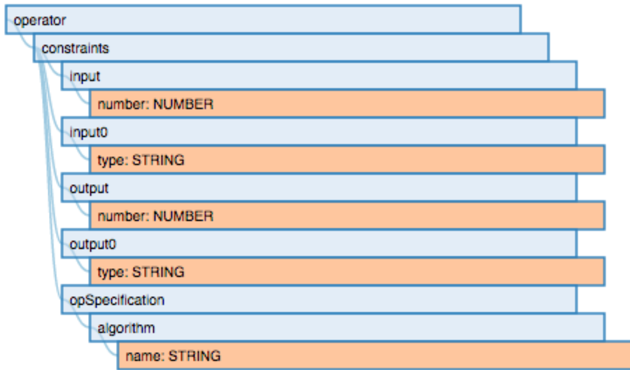

Figure 2: The generic metadata tree for operator

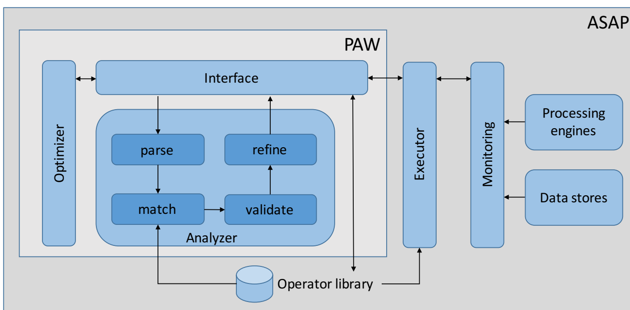

Figure 3: The architecture of PAW

of data. Each vertex in a workflow represents *one or more* tasks of data processing. Each task is a set of *inputs*, *outputs* and an *operator*. Tasks may share or not inputs, but they do not share operators and outputs. The inputs and outputs of the tasks of a vertex can be related to incoming and outgoing edges of this vertex, but they do not identify with edges: inputs and outputs represent consumption and production of data, respectively, and edges represent the flow of data. Figure 1 displays a workflow about a product marketing campaign. It combines sales data with sentiments about the product, gleaned from tweets crawled from the Web.

Data and operators can be either abstract or materialized. Abstract are the operators and datasets that are described partially or at a high level by the user, when composing the workflow, whereas materialized are the actual operator implementations and existing datasets, either provided by the user or residing in a repository. Both 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, location of data objects or operator invocation scripts, data schemas, implementation details, engines etc. These metadata are used to:

- Match abstract operators to materialized ones.
- Check the usage of a dataset as input for an operator. If the dataset does not match the operator's input, its metadata can be also used to check for appropriate transform/move operators that can be applied.
- Provide optimization parameters, e.g. profiling of input/output.
- Provide execution parameters like a file path or arguments for the execution of the operator script.

The internal representation of a workflow is in the Tree-metadata language, which captures structural information, operator properties (e.g., type, data schemas, statistics, engine and implementation details, physical characteristics like memory budget), and so on. The metadata tree is user extensible. Figure 2 shows the generic metadata tree for an operator. To allow for extensibility, the first levels of the metadata tree are predefined. Users can add their ad-hoc subtrees to define their custom data or operators. Moreover, some fields (like the ones related operators and data) are compulsory, while the rest are optional and user-defined. Materialized data and operators need to have all their compulsory fields filled in with information. Abstract data and operators do not adhere to this rule.

3. ARCHITECTURE & IMPLEMENTATION

PAW is part of a larger system, called Adaptable Scalable Analytics Platform (ASAP) [1], 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 of data stores and processing engines. Figure 3 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:

- **Operator library.** This library shows operator implementations imported from the ASAP library. The operators are classified as, either analytics 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 interface enables users to interactively create and/or modify a workflow.
- **Analyzer.** The analyzer parses the workflow, identifies operators and data stores and maps them to the library of operators, generates metadata of edges, finds edges where the data conversion should be applied and adds the appropriate conversions.
- **Optimizer.** The optimizer generates a functionally equivalent workflow, optimized for performance objective.
- **Executor.** The executor receives workflows from the optimizer and schedules them for execution. When a workflow is ready for execution, it dispatches the workflow to the engines and monitors its execution.

Code generation and the executor is implemented in Java. The interface is a web application in Jade [6] and CoffeeScript [2], and Grunt [3] compiles it in HTML and JavaScript, respectively. The interface communicates with other modules using Nginx web server [9] and PHP-FPM [11]. The analysis and optimisation modules are implemented in Python.

4. FUNCTIONALITY OF PAW

This section describes the PAW functionalities and discusses relevant aspects of the components.

4.1 Management of operators

Each operator can have an abstract definition and several implementations, i.e. one or more implementations per engine. For example, a 'join' of two inputs, has an abstract definition, and can be implemented for a relational DBMS

Operator	Blocking	Non-blocking	Restrictive
Filter		x	x
Calc		x	
Filter Join	x		
groupBy Sort	x		
PeakDetection	x		
Tf-idf	x		
k-Means	x		

Table 1: Categorization of operators

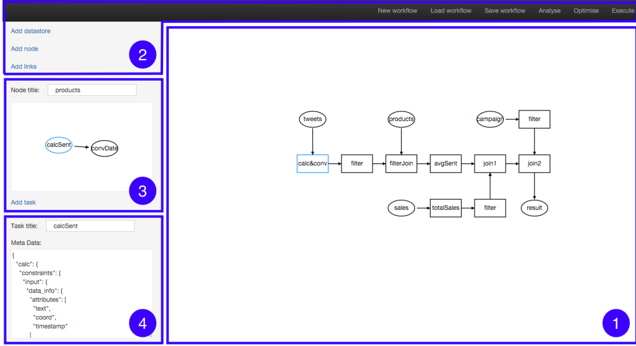

Figure 4: Interface of PAW

and a NoSQL database. An operator that performs a simple operation or a complex algorithm computation needs to have a tailored implementation for every engine on which it is going to be executed. An operator definition includes restrictions on the type and number of inputs and specifies the number and type of outputs.

Operators are categorized as:

- **Blocking operators** require knowledge of the whole data, e.g., a grouping operator or an operator *join* or *sort*.
- **Non-blocking operators** process each tuple separately, e.g., operators *filter* or *calc*[1].
- **Restrictive operators** output a smaller data volume than the incoming data volume, e.g. *filter*.

Defined and implemented operators form a library from which a user can select operators to describe tasks. Table 1 shows operators from the library and their categorization.

Users can register their own operators, provide respective implementations and define optional attributes of a new operator. These attributes include functions to compute cardinality and processing cost, and characteristics of the operator. In most cases, the operator developer or provider does not disclose a cost formula for the operator. Then, PAW can use the ASAP profiling process for operators using micro-benchmarks. As an optional step, PAW allows users to run their workflows with a data sample and uses the obtained statistics to fine-tune our cost models before workflow optimization.

4.2 Design of a workflow

A workflow is created in the PAW interface, which consists of several areas (Figure 4) that perform the following:

- Display the workflow (Area 1).
- Design a new workflow adding vertices and edges, save and load it (Area 2).
- Perform workflow analysis or optimization (Area 2).
- Add tasks from a library or create new ones (Area 3). A task from the library is accompanied by a set of metadata. A task created from scratch has a metadata tree with predefined first levels; users can add their ad-hoc subtrees to define their custom data or operators.
- Display metadata of the selected task (Area 4).

[1] *calc* is a generic operator that can be customized for a specific numeric calculation

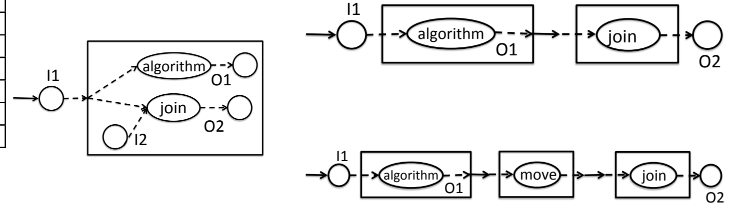

Figure 5: Analysed workflows for a multi-task vertex

4.3 Analysis of a workflow

The workflow model alleviates from the user the burden of determining any or some execution semantics of the application logic. The execution semantics of the workflow includes the execution of tasks in vertices and the execution of input-output dependencies of edges. The determination of the execution semantics of vertices and edges leads to an execution plan of the workflow. We refer to this plan as the *analysed* workflow. The latter is actually an enhancement of the initial workflow with more vertices and substitution of vertices and/or edges in the initial workflow with others.

PAW analyses a workflow in several steps:

1. Parses the workflow.
2. Categorizes operators (see Section 4.1).
3. Validates consistency. A workflow is checked for cycles and correspondence of metadata of adjacent vertices. Cycles cannot be resolved, the analysis stops and returns a list of errors. If possible, metadata mismatches are solved by adding associative tasks in Step 6.
4. Generates metadata of edges, as a join of input and output metadata of source and target vertices, respectively.
5. Splits multi-task vertices to several single-task vertices: A vertex that corresponds to multiple tasks is replaced with an *associative subgraph* that contains a set of new vertices that correspond to these tasks. New vertices may correspond 1-1 to tasks, but it can be the case that two or more vertices correspond to the same task (task replication).
6. Augments the workflow with associative tasks that are converting data flow: buffers and format conversions.

Users may describe the same application logic by creating workflows with different levels of detail concerning the execution planning of this logic.The analysis phase determines missing execution semantics. Figure 5 shows an example: A user defines a vertex with two tasks, algorithmic processing on some data, and a join of these with some other data. The user is not interested or does not know the execution details of this complex task. This representation depicts that the two tasks should be executed together, after the tasks of the vertices on which this vertex depends, and before the tasks of vertices that depend on this vertex. Another user, represents the same tasks with two connected vertices, dictating that the join should be executed on the data processed first by the algorithm. A third user dictates even more detail in the execution plan, by adding one more vertex that includes a task that moves the data, e.g. from one disk to another.

4.4 Optimization of a workflow

After the analysis phase, a workflow is optimized for performance. The optimization uses the following operations:

- **Swap.** The *swap* operation applies to a pair of vertices, v_1 and v_2, which occur in adjacent positions in an 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.
- **Merge.** The *merge* operation takes as input two vertices and produces one new vertex that includes the tasks of both initial vertices. The latter may either be connected with an edge, i.e. there is some task dependency(ies), or not. The goal of *merge* is to allow for a united optimisation of the tasks included in the two vertices, e.g. joint micro-optimization on an execution engine.
- **Split.** The *split* operation takes as input one vertex and produces two new vertices that, together, include all the tasks included in the initial vertex. The new vertices may or may not be connected. The goal of *split* is to lead to separate optimisation of subgroups of the tasks.

The Optimizer applies to the analysed workflow a series of the above operations, each producing a functionally equivalent workflow with possibly different cost. The goal is to find an optimal workflow in the state space of equivalent workflows (for the optimization algorithm see [16]). To improve search, the space is pruned employing heuristics:

- **H**1: Move restrictive operators to the root of the workflow to reduce the data volume, e.g., rather than *extract* →*function* →*filter* do *extract* →*filter* →*function*.
- **H**2: Place non-blocking operators together and separately from blocking operators, require knowledge of the whole dataset, e.g., rather than *filter* →*sort* →*function* →*group* do *filter* →*function* →*sort* →*group*.
- **H**3: Parallelize adjacent non-blocking operators so that they can be executed concurrently on separate processors, e.g., split *filter1* →*filter2* to two new parallel paths. Parallelized workflow parts should be chosen such that their latency is approximately equal.

5. DEMONSTRATION

In the following, we describe the demonstration of PAW.

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 [4], Spark [15] and Weka [18]. The distribution of Hadoop is CDH 4.6.0 and the version of Spark is 1.4.1.

Workloads. The demonstration uses synthetic and real workflows on real data. The synthetic workflows are constructed based on ETL benchmarking [12]. Real workflows and data come from the two use cases of ASAP [1] and belong to the domains of telecommunications and web analytics. The telecommunication use case involves processing anonymised Call Detail Records (CDR) data for Rome, from 01/06/2015 until 30/06/2015 and stored in CSV format. For the computation on graph-structured data workflows are implemented in Apache Spark.

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 1 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 aims to exhibit the functionalities of PAW, focusing on the following aspects: (a) the usability of the platform for workflow creation by users that have different expertise, (b) the effectiveness of the automated analysis of the execution of the workflow, and (c) the effectiveness of the automated optimization in workflow execution. The demonstration includes interesting scenarios for all (a,b,c) to be shown to the audience, but also interactive scenarios, especially for (a) and (b), which allow the audience to experience the functionalities of PAW. The interactive scenarios enable the participant to 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. Concerning (a) and (b), the scenarios exhibit how the same application logic can be expressed via workflow versions that have different level of detail of execution semantics, and how the analysis phase specifies missing execution semantics through already executed workflows and/or by giving alternative choices to the user. Concerning (c), the scenarios show how new operators are added, including cost functions, and how the latter may be tuned by running a workflow with sampling for statistics collection; finally, the scenarios show actual workflow execution and the optimization benefit and tradeoffs on different engines.

Acknowledgments

This work has received funding from the European Union's 7th Framework Programme under grant agreement n° 619706.

6. REFERENCES

[1] Asap. http://www.asap-fp7.eu/.
[2] Coffeescript. http://coffeescript.org/.
[3] Grunt - the javascript task runner. http://gruntjs.com/.
[4] Apache hadoop. http://hadoop.apache.org/.
[5] Informatica 'powercenter'. http://www.informatica.com/products/powercenter/.
[6] Jade - template engine. http://jade-lang.com/.
[7] V. Kantere and M. Filatov. A framework for big data analytics. In *C3S2E*, 2015.
[8] V. Kantere and F. Maxim. Modelling processes of big data analytics. In *WISE (To Appear)*, 2015.
[9] Nginx. http://nginx.org/.
[10] Oracle warehouse builder 10g. http://www.oracle.com/technology/products/warehouse/.
[11] Php-fpm (fastcgi process manager). http://php-fpm.org/.
[12] A. Simitsis, P. Vassiliadis, U. Dayal, A. Karagiannis, and V. Tziovara. Benchmarking ETL workflows. TPCTC, 2009.
[13] A. Simitsis, K. Wilkinson, M. Castellanos, and U. Dayal. Optimizing analytic data flows for multiple execution engines. In *ACM SIGMOD*, 2012.
[14] A. Simitsis, K. Wilkinson, U. Dayal, and M. Hsu. Hfms: Managing the lifecycle and complexity of hybrid analytic data flows. In *ICDE*, 2013.
[15] Apache spark. https://spark.apache.org/.
[16] Technical report. https://github.com/project-asap/workflow/blob/master/tech_report/report.pdf.
[17] W. M. P. Van Der Aalst and A. H. M. T. Hofstede. Yawl: Yet another workflow language. *Inf. Syst.*, 30(4), June 2005.
[18] Weka. http://weka.pentaho.com/.