Optimizing, Planning and Executing Analytics Workflows over Multiple Engines

Katerina Doka ⋆, Maxim Filatov #, Victor Giannakouris ⋆, Verena Kantere #, Nectarios Koziris ⋆, Christos Mantas ⋆, Nikolaos Papailiou ⋆, Vassilis Papaioannou ⋆, Dimitrios Tsoumakos ◊

⋆ *National Technical University of Athens, Greece*
University of Geneva, Switzerland
◊ *Ionian University, Greece*

ABSTRACT

Big data analytics have become a necessity to businesses worldwide. The complexity of the tasks they execute is ever increasing due to the surge in data and task heterogeneity. Current analytics platforms, while successful in harnessing multiple aspects of this "data deluge", bind their efficacy to a single data and compute model and often depend on proprietary systems. However, no single execution engine is suitable for all types of computation and no single data store is suitable for all types of data. To this end, we present and demonstrate a platform that designs, optimizes, plans and executes complex analytics workflows over multiple engines. Our system enables users to create workflows of variable detail concerning the execution semantics, depending on their level of expertise and interest. The workflows are then analysed in order to determine missing execution semantics. Through the modelling of the cost and performance of the required tasks over the available platforms, the system is able to match distinct workflow parts to the execution and/or storage engine among the available ones in order to optimize with respect to a user-defined policy.

1. INTRODUCTION

Big data analytics have become indispensable for the majority of industries, enabling engineers, analytics experts and scientists alike to tap the potential of vast amounts of business-critical data. Such data analysis demands a high degree of parallelism in both storage and computation and has given rise to diverse execution engines and data stores that target specific data and computation types. Performance optimizations thereof assume strictly single-engine environments, thus considering specific data formats and query/analytics task types [9, 14].

However, modern workflows have become increasingly long and complex and may include multiple data types such as relational, key-value, graph, etc., as well as greatly diverse operators, ranging from simple Select-Project-Join (SPJ) and data movement to complex NLP-, graph- or custom business-related operations. What is more, they are executed under varying constraints and policies (e.g., optimize performance or cost, etc.). There currently exists no single platform that can optimize for this complexity [17].

Sensing this trend, companies now offer cloud software distributions that incorporate different processing frameworks, data stores and libraries to alleviate the burden of multiple installations and configurations [1, 2]. Yet, such multi-engine environments lack a *meta-scheduler* that could automatically match tasks to the right engine(s) according to multiple criteria, deploy and run them without manual intervention.

To address multi-engine analytics workflow optimization we demonstrate an integrated, open source platform for creating, managing, executing and monitoring complex analytics workflows[1]. Its goal is twofold: (a) To allow the user to design workflows that span multiple engines and data stores by either giving specific details on execution semantics of tasks and data stores or leaving the platform to determine them through an automated workflow analysis phase. Then, the workflow goes through an automated optimization phase, before being sent for execution. (b) To provide cost-based and customizable resource management of the diverse execution and storage engines available by adaptively choosing to execute each sub-part of the workflow to a (possibly different) deployed engine in order to enhance performance

Towards this direction, our system supports the most prevalent open-source execution models (e.g., Map-Reduce, Bulk Synchronous Parallel) as well as state-of-the-art centralized and distributed storage engines (RDBMSs, NoSQL, distributed file-systems, etc.) and is able to optimize workflows consisting of tasks that range from simple group-by, aggregation or complex joins between different data sources to machine-learning tasks.

Our demonstration will showcase our system's ability to i) model operator performance according to different engines and their resources and ii) adaptively decide on which operator version to run based on the optimization policy and the available engines. The demonstration platform will integrate Hadoop [4], Spark [5], PostgreSQL [3] and HDFS and operate upon real-life and synthetic workflows chosen to include diverse datasets and computation types. The participants will have a rich interaction with the platform, controlling

[1]The presented system is part of the *ASAP (Adaptive, highly Scalable Analytics Platform)* EU-funded project. ASAP envisions a unified, open-source execution framework for scalable data analytics. http://www.asap-fp7.eu/

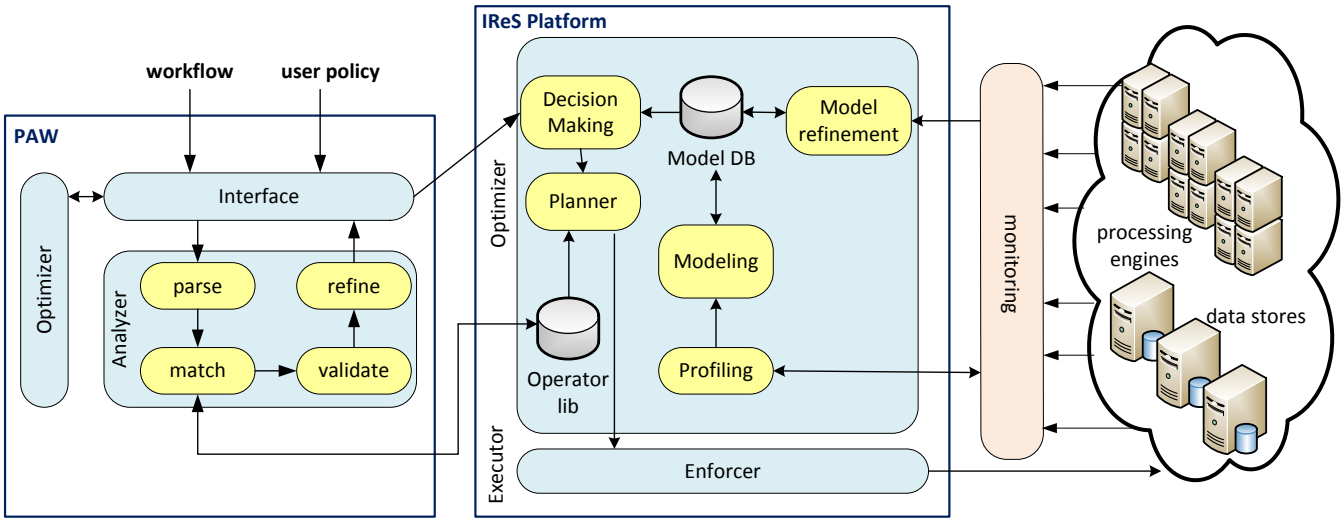

Figure 1: System architecture

policy and input aspects, while being able to evaluate the advantages of multi-engine optimization by inspecting generated plans and output.

2. SYSTEM ARCHITECTURE

The system integrates two core components of the ASAP project, namely *PAW* (Platform for Analytics Workflows), which acts as the workflow management tool of ASAP and *IReS* (Intelligent Resource Scheduler) [7], which creates the optimal plan and executes it. The two components collaborate with the IReS platform for the creation, optimization and execution of user workflows. Figure 1 depicts the architecture of the integrated system to be demonstrated.

The components of PAW communicate using the internal workflow representation and are:

- **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 existing operator implementations, generates metadata of edges, finds edges where the data conversion should be applied and adds the appropriate conversions. The operator implementations reside in the IReS operator 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.
- **Optimizer.** The optimizer generates a functionally equivalent workflow, optimized for performance objective.

After the generation of the optimized workflow, PAW hands it to the IReS platform for further cost-based optimization according to the use-defined policy, planning and execution. IReS receives from PAW the input that is necessary for its operations and identifies execution artifacts such as operators, data, their dependencies and accompanying metadata. Moreover, it validates the user-defined policy. All this information must be robustly identified, structured in a dependency graph and stored.

IReS is decomposed in two layers, the *cost-based optimizer* and the *executor* layer:

- The **cost-based optimizer layer** is responsible for optimizing the execution of an analytics workflow with respect to the policy provided by the user. The core component of the optimizer is the *Decision Making* module, which determines the optimal execution plan in real-time. This entails deciding on where each subtask is to be run, under what amount of resources provisioned, the plan for moving data to/from their current locations and between runtimes (if more than one is chosen) and defining the output destinations. Such a decision must rely on the characteristics of the analytics task in hand and the models of all possible engines. These models are produced by the *Modeling* module and stored in a database called *Model DB*. The initial model of an engine results from profiling and benchmarking operations in an offline manner, through the *Profiling* module. This module directly interacts with the pool of physical resources and the monitoring layer in-between. While the workflow is being executed, the initial models are refined in an online manner by the *Model refinement* module, using monitoring information of the actual run. Such monitoring information is kept in the IReS DB and is utilized by the decision making module as well, to enable real-time, dynamic adjustments of the execution plan based on the most up-to-date knowledge.
- The **executor layer** is the layer that enforces the optimal plan over the physical infrastructure. It includes methods and tools that translate high level "start runtime under x amount of resources", "move data from site Y to Z" type of commands to a workflow of primitives as understood by the specific runtimes and storage engines. Moreover, it is responsible for ensuring fault tolerance and robustness through real-time monitoring.

Both PAW and IReS are open source[2]. The interface of PAW is a web application in Jade [10] and CoffeeScript [6], and Grunt [8] compiles it in HTML and JavaScript, respectively. It communicates with other modules using Nginx web server [15] and PHP-FPM [16]. The analysis and optimisation modules of PAW are implemented in Python. The IReS platform modules are implemented in Java. The en-

[2]https://github.com/project-asap

forcer module relies on YARN [18] and extends Cloudera Kitten [13], a framework that allows the definition of operator execution on top of YARN.

3. DEMONSTRATION DESCRIPTION

The demonstration shows how PAW and IReS collaborate in order to optimize an analytics workflow, produce an execution plan that conforms to a user-defined policy and enforce it. We show that the system can be used by users with different level of expertise in order to create workflows. The tool implements a novel workflow language [11, 12] 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. The analysed workflow then goes through a decision making phase, where the execution plan that optimizes user-defined parameters (e.g., performance, cost, etc.) is produced based on the operator models. Finally, the optimized workflow is sent for execution.

The system is controlled by a comprehensive web-based GUI that attendees will utilize. The GUI controls a cloud-based deployment of several runtime engines and data stores over 16 virtual machines of an Openstack cluster.

The users will be able to test the system either using one of our predefined workflows or assembling their own, using operators from the ASAP operator library. A diverse set of operations of varying complexity is covered including basic SQL queries (selections, projections, joins) and ML algorithms (classification and clustering).

The predefined workflows represent real use cases driven by business needs. These cover complex data manipulations in the areas of business analytics on telecommunication data and web data analytics, provided by a large telecommunications company and a well-known web archiving organization respectively. The input datasets for these workflows consist of anonymized telecommunication traces and web content data (WARC files). Subsets of those datasets can be used for each of the available workflows.

4. USE CASE EXAMPLE

This example shows the telecommunication use case of ASAP. This workflow detects peaks representing an event, by comparing the density of population within a region in a given moment against the expected density for that area at that hour of the day. It involves processing of anonymized Call Detail Records (CDR) data (residing in an RDBMS) via clustering along time and space in order to detect peaks in load according to a set of criteria. The dataset of peaks is used to discover clusters of calls that occur with or without regularity. Figure 2 displays the workflow.

The first steps of the process, *data prep* and *convert ts*, consist of call aggregation by user_id and extraction of time periods (for instance, days) and smaller time slots (for instance, hours). The *regions* table contains links of tower ids and regions. Time slots and regions of analysis, which are used in *filter region*, are parameters provided by the user. These two parameters, then, allow defining a spatio-temporal grid, and each observation of an input dataset can be assigned to one of its cells. The number of observations that fall in a cell defines its density. The input data is partitioned into two sets: a training dataset (for instance, 1st month) and a test dataset (for instance, left time period), two *filter*s following after *convert ts* produce them. For both datasets the spatiotemporal grid of densities is computed. The first is used to compute the densities of a typical period for each region. The second dataset is then compared against such typical period in order to detect significant deviations. Based on the densities obtained for each region and each time slot over the training dataset, an expected density value is computed for each region, by averaging the densities (*week aggr*ation) measured at the same time slot of all the periods in the time window covered by the dataset. For instance, we might obtain an expected density for each pair (region, hour of the day), i.e., 24 values for each region, assuming 24 one-hour time-slots. Then, for each region and each time-slot, the corresponding density is compared against its expected value (*join* and *calc ration*): if the difference is significant (*filter peaks*), an event of form (region, weight, time slot) is produced (*events*), representing its spatiotemporal slot and a discretized measure (weight) of how strong was the deviation.

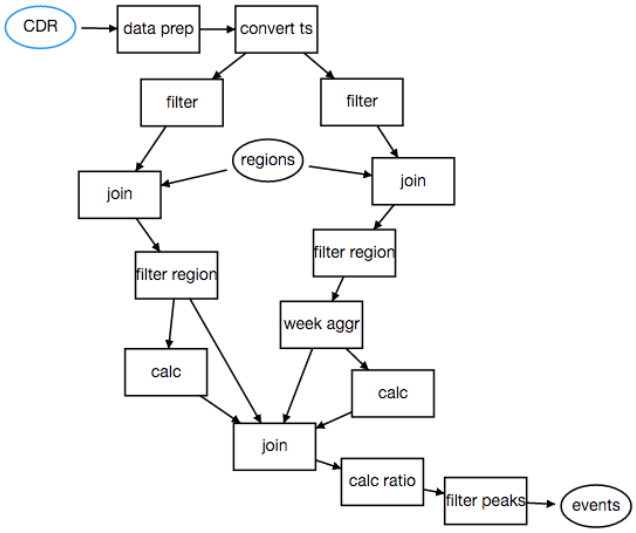

Figure 2: Workflow for the detection of peaks

Figure 3 displays the optimized workflow produced by PAW's Optimizer. In this there are tasks pushed closer to the start of the workflow and merged in a single node, these tasks are in the original nodes *join*, *filter region* and *regions* dataset. This optimization has been done accordingly to a heuristic that moves restrictive operators (in this case *filter region*) to the root of the workflow. Furthermore the Optimizer of PAW checks that this rearrangement and merging can be performed.

This workflow is then handed to the IReS platform, where the conceptual workflow operators are matched with implementations that exist in the operator library, creating all alternative execution paths. After consulting the cost models of the implemented operators, the platform chooses the plan and decides on the exact engines and their setup in order to optimize the workflow in terms of the user-defined policy. In our running example, if the input dataset is large the decision making module opts for a plan that contains distributed implementations of the operators in Spark rather

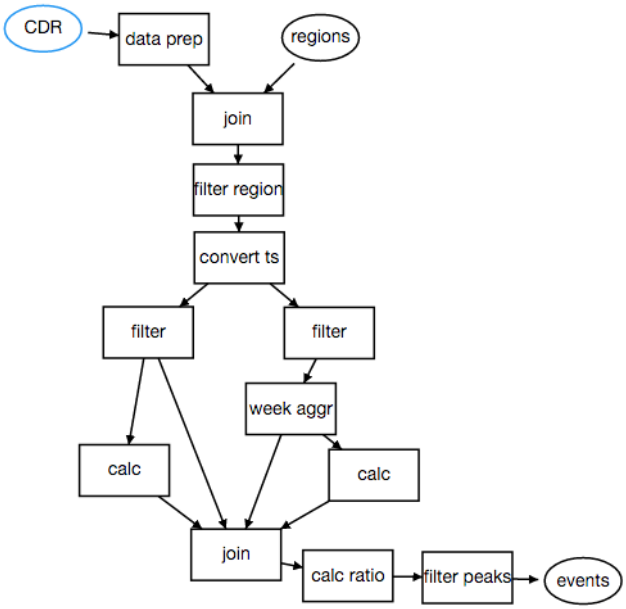

Figure 3: Optimised workflow for the detection of peaks

than centralized ones in PostgreSQL. This choice entails the transfer of the dataset from PostgreSQL (where it originally resides) to HDFS, thus an auxiliary *move* operator is added to the workflow. Moreover, assuming the user only cares to minimize execution time, IReS decides to utilize all available nodes of the cluster.

Acknowledgements

This work has been supported by the European Commission in terms of the ASAP FP7 ICT Project under grant agreement n^o 619706. Nikolaos Papailiou has received funding from IKY fellowships of excellence for postgraduate studies in Greece - SIEMENS program.

5. REFERENCES

[1] Cloudera Distribution CDH 5.2.0. http://www.cloudera.com/content/cloudera/en/downloads/cdh/cdh-5-2-0.html.

[2] Hortonworks Sandbox 2.1. http://hortonworks.com/products/hortonworks-sandbox/.

[3] Postgresql. http://www.postgresql.org/.

[4] The Apache Software Foundation: Apache Hadoop. http://hadoop.apache.org/.

[5] The Apache Software Foundation: Apache Spark. https://spark.apache.org/.

[6] Coffeescript. http://coffeescript.org/.

[7] K. Doka, N. Papailiou, D. Tsoumakos, C. Mantas, and N. Koziris. Ires: Intelligent, multi-engine resource scheduler for big data analytics workflows. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1451–1456. ACM, 2015.

[8] Grunt - the javascript task runner. http://gruntjs.com/.

[9] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A Self-tuning System for Big Data Analytics. In *CIDR*, volume 11, pages 261–272, 2011.

[10] Jade - template engine. http://jade-lang.com/.

[11] V. Kantere and M. Filatov. A framework for big data analytics. In *C3S2E*, 2015.

[12] V. Kantere and F. Maxim. Modelling processes of big data analytics. In *WISE*, 2015.

[13] Cloudera kitten. https://github.com/cloudera/kitten.

[14] H. Lim, H. Herodotou, and S. Babu. Stubby: A Transformation-based Optimizer for Mapreduce Workflows. *Proceedings of the VLDB Endowment*, 5(11):1196–1207, 2012.

[15] Nginx. http://nginx.org/.

[16] Php-fpm (fastcgi process manager). http://php-fpm.org/.

[17] D. Tsoumakos and C. Mantas. The Case for Multi-Engine Data Analytics. In *Euro-Par 2013: Parallel Processing Workshops*, pages 406–415. Springer, 2014.

[18] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.