FP7 Project ASAP

Adaptable Scalable Analytics Platform

ASAP D5.4
Dynamic re-calibration of processing

WP 5 – Adaptive Data Analytics

Nature: Report

Dissemination: Public

Version History

Version	Date	Author	Comments
0.1	7 Feb 2017	V. Kantere, M. Filatov	Initial Version
0.2	20 Feb 2017	V. Kantere, M. Filatov	First Revision
1.0	1 Mar 2017	V. Kantere, M. Filatov	Final Version

Acknowledgement This project has received funding from the European Union's 7th Framework Programme for research, technological development and demonstration under grant agreement number 619706.

Executive Summary

This deliverable is a report on the recalibration module of the Platform of Analytics Workflows (PAW). This module enables the analytics expert to change the original task or workflow by altering the task parameters or infusing new tasks, while he monitors the progress of processing in terms of data accessing and resource utilization based on input from the runtime machines or the visualization tool. The report first gives a quick overview of the PAW architecture, then delves into the details of recalibration techniques and demonstrates the efficiency of their application on real-world applications, as well as on a synthetic workflows.

Contents

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. Workflow execution can be extremely resource- and time-consuming. Thus, a system that enables such long-term analytics processes on Big Data needs to be able to show the progress of the execution and the intermediate results. This means that the user should be able to monitor which workflow tasks have been executed, their produced results, which tasks are currently executing, as well as data accessing and resource utilization based on input from the runtime machines or the visualization tool. Further, the system should allow the user to influence workflow processing. This means that the system should provide methods that enable the analytics expert to change a workflow by altering task parameters or infusing new tasks manually at runtime, or even to predefine automatic changes at workflow creation by providing alternative workflow branches. Such *recalibration* methods constitute a powerful functionality of a workflow management system, since they enable the gradual design of exploratory analytics workflows based on feedback from intermediate results, as well as the efficient error handling of complex and long-running workflows.

In this deliverable we focus on the novel functionality of PAW (Platform for Analytics Workflows)[1] for workflow recalibration. PAW is a platform for the design, management, analysis, optimization and execution of analytics workflows. The first version of PAW is presented in [1, 2, 3] and includes the functionalities of workflow design and analysis in order to clarify execution semantics, single workflow optimization and multi-workflow optimization. In this deliverable we present for the first time the new functionality of PAW on workflow recalibration. It includes novel techniques for (a) manually changing a workflow at runtime and re-executing it avoiding repeated computations, called *recovery and monitoring points* technique (3.1); (b) automatically changing a workflow at runtime based on conditional structures *if-then-else* (3.2) and *goto* statements (3.3).

1.1 Task Description

This deliverable describes work performed in task T5.3 “Dynamic re-calibration of processing” by UNIGE. The task aims to produce a module that implements the re-calibration of processing, that enables the analytics expert to change the original task or workflow by altering the task parameters or infusing new tasks, while he monitors the progress of processing in terms of data accessing and resource utilization based on input from the runtime machines or the visualization tool. During Y3 of the ASAP project, the system for workflow management and optimization produced during the second year and tasks T5.1-5.2 was augmented with a recalibration module.

[1] In previous deliverables, this tool was called WMT (Workflow Management Tool).

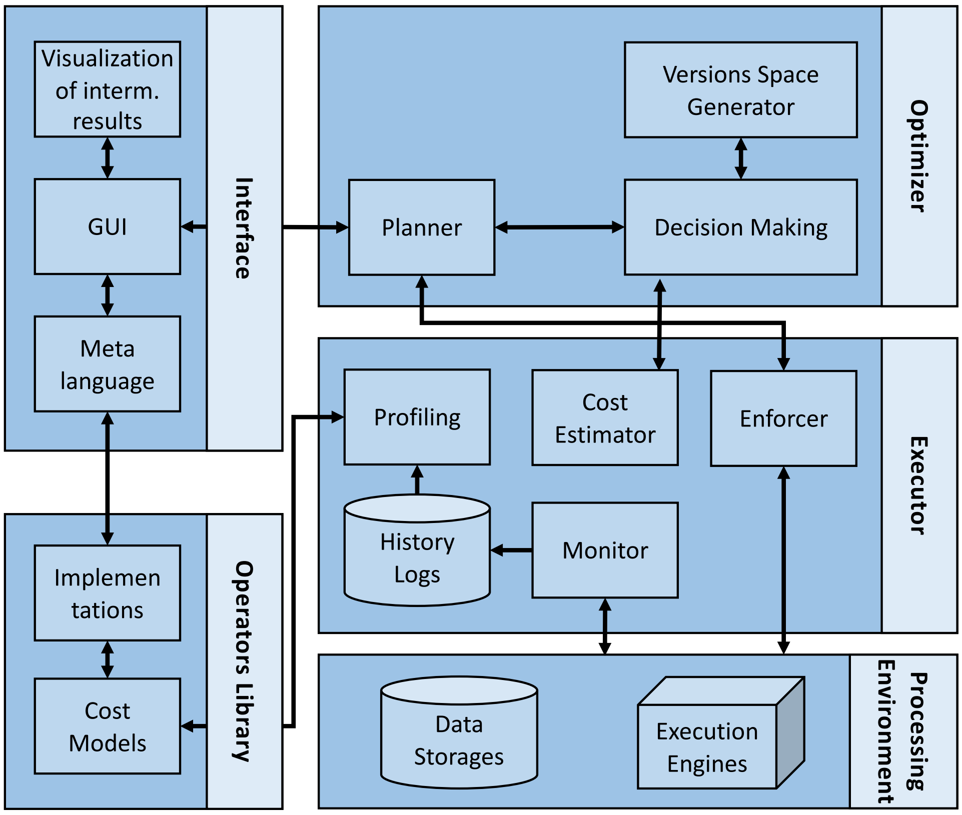

Figure 1: The architecture of PAW and its interaction with IRES

1.2 Overview of PAW

PAW implements a novel workflow model [5, 6]. A workflow W is a directed, acyclic graph (DAG) $G = (\mathcal{V}, \mathcal{E})$. The vertices $\mathcal{V}$ represent data processing tasks $\mathcal{T}$ and the edges $\mathcal{E}$ 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. PAW is a part of the system 'Adaptable Scalable Analytics Platform' (ASAP) [7], but it can also stand as an independent tool for workflow management and optimization. PAW enables workflow design by users with various expertise, the automation of workflow analysis in order to clarify and specify execution semantics, single and multiple workflow optimization with respect to time efficiency, over a diverse collection of data stores and processing engines, monitoring of workflow execution and manual and automatic workflow recalibration. Figure 1 depicts the architecture of PAW, as well as its interaction with the rest of ASAP. PAW consists of four layers: Operators Library, Interface, Optimizer, and Executor. These provide for workflow design, optimization, and execution dispatch, respectively. Workflows are executed on a set of *execution engines* and *storage repositories* of the multi-engine

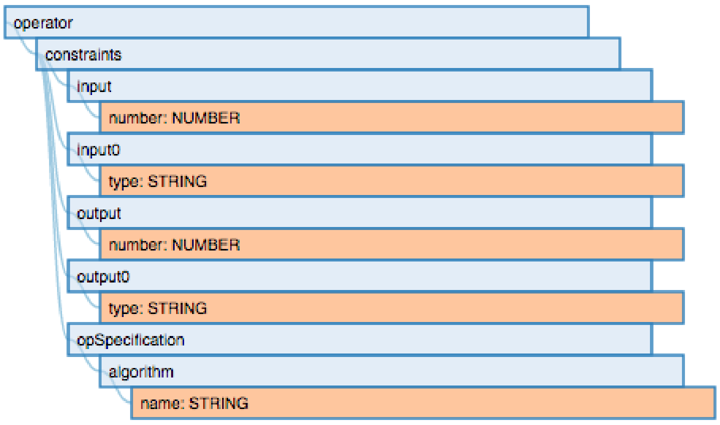

Figure 2: The generic metadata tree for operator

environment.

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. The recalibration module has supplemented the library with several operators, called recalibration points and described further in Section 3.

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 (e.g., type, data schemas, statistics, engine and implementation details, physical characteristics like memory budget), and so on. The metadata tree is user extensible. 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. Figure 2 shows the generic metadata tree for an operator. Furthermore the interface allows users to observe the process of execution and intermediate results of a workflow for the recalibration needs.

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, which returns 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. As an execution system, PAW uses IReS [8].

1.3 Purpose of the document

This document serves as a report on the recalibration module of PAW and accompanies its prototype implementation. Its purpose is to delve into the details of recalibration techniques and showcase their application. This includes detailed description of each novel recalibration technique, as well as the description of the overall recalibration process. Finally, we demonstrate the recalibration module of PAW on specific use-cases from D9.2 [9] and D8.2 [10] and on a set of synthetic workloads.

1.4 Document structure

The rest of this document is structured as follows:

- Chapter 2 gives a brief overview of the workflow model and states the problem of workflow recalibration. Moreover, it gives a motivating example driven by the use-case scenario of D9.2 [9].
- Chapter 3 presents recalibration techniques.
- Chapter 4 showcases the application of the recalibration process describes on both synthetic workflows and workflows of real-world applications.
- Chapter 5 summarizes related work in the topic of the workflow recalibration.
- Chapter 6 concludes the deliverable.

2 Problem Discussion

2.1 Motivating example

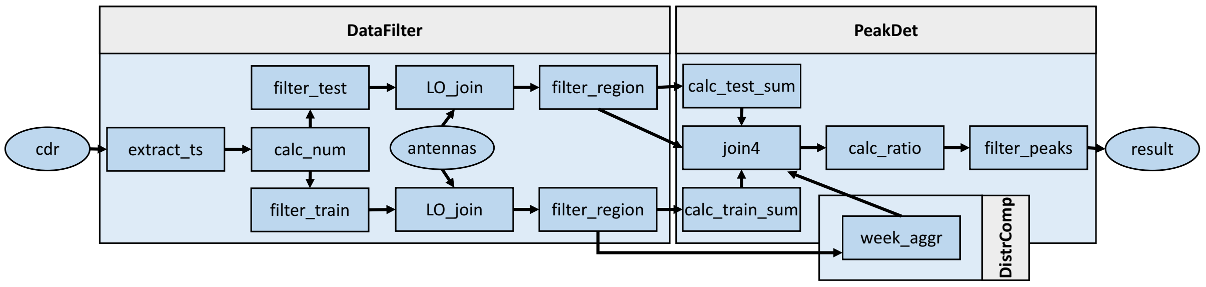

Figure 3: 'Peak Detection' of mobile calls workflow

Figure 3 shows a real-world analytics workflow from WIND, which involves processing of the anonymised Call Data Records (CDR), collected in Rome for 2015 year and stored in HDFS, to populate a report on a dashboard. The report lists peaks in calls and their ratios to an averaged number of calls over a training period (one month). Peaks are defined by "differences from typical". The workflow extracts the day of the week, hour of the day from timestamp for each call record (*extract_ts*). The task *calc_num* sums calls at one-hour intervals. Then, two filters split the data to training and test datasets. Further, analysis is limited to specific geographical regions and, then, the number of calls in the training period is averaged over each mobile tower region, day in a week and hour in a day (*week_aggr*); this is the typical distribution of calls. Next, *calc_test_sum* and *calc_train_sum* produce sum of calls in each day of the test and training datasets. Then, test and training data are joined and the ratio of calls to average number is produced. The *filter_peaks* finds ratios that are over a specific limit. These peaks is the sought information.

Initially this workflow comprised three complex UDFs, (*DataFilter*, *DistrComp* and *PeakDet*), implemented in PySpark. Later, for optimization needs [14] they were decomposed to smaller basic tasks, the operators of which have implementations in Spark and PostgreSQL. It is quite common that industrial workflows, like this one, are versioned and updated with time, resulting in a design that may not be optimal for the exploration procedure that the analytics expert needs to follow. In this example, the expert needs to explore the peaks one by one. This requires a complete restart of the workflow with changed search regions, a parameter of the *filter_region* tasks. This problem can be solved with recalibration methods that allow for the re-usage of the intermediate results produced by the tasks leading to the *filter_region* tasks, without re-executing the first. Also, recalibration methods would enable the expert to monitor the result of *filter_region*, visualized on a geographical map, so that he can observe faster call congestion and decide to change the search region. Furthermore, methods for automatic recalibration can enable the expert to predefine conditions on intermediate results and, also, predefine decisions to be taken according to the outcome of condition evaluation.

2.2 Problem statement

In this section we briefly recall the context and used concepts. These are thoroughly described in previous deliverables D5.1 [12] and D5.2 [3].Then we state the problem of recalibration.

We assume a data processing environment that comprises numerous machines (e.g. a cluster) on which a variety of processing engines are installed (e.g. traditional and modern DBMSs). Each engine has access to a local data store. This environment may include multiple instances of the same engine (e.g. PostgreSQL). The processing environment takes as input logical workflows of data processing on a set of input data $\mathcal{D}_i$, producing a set of output data $\mathcal{D}_o$. The logical workflow (hereafter workflow) W is directed acyclic graph (DAG), $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ the set of edges of the graph G. Therefore, $W = \{G, \mathcal{D}_i, \mathcal{D}_o\}$.

Each vertex $v \in \mathcal{V}$ represents a logical processing task t and each edge $e \in \mathcal{E}, e = (v_1, v_2)$ the flow of data between two vertices v_1 and v_2. Therefore, a task of v_1 processes the data output by other tasks of v_1, for which there exists edges $e = (v_1, v_2)$ in G. Each task is a set of *inputs*, *outputs* and an *operator*. The operator of any task t is accompanied by a set of parameters $\mathcal{P}_t$.

The *recalibration* of a workflow entails the following requirements:

Enable access to intermediate results. The intermediate result of workflow execution is the (complete) result of a task t that has been executed, while the execution of the entire workflow is not yet completed. The user can have access to the intermediate results of executed tasks their visualization.

Enable workflow changes at runtime. Such workflow changes consist of (a) altering the task parameters $P_t \rightarrow P'_t$ of a workflow task t and (b) changing a workflow graph $G = (\mathcal{V}, \mathcal{E}) \rightarrow G' = (\mathcal{V}', \mathcal{E}')$, that includes infusing new tasks and/or removing existing ones.

Avoid repeated computations. Workflow changes at runtime need to incur re-execution of the workflow. However, we require that only tasks of changed workflow parts are re-executed, thus avoiding to repeat the computation of tasks the intermediate results of which do not change after the recalibration of the workflow.

In the following we describe the proposed novel recalibration techniques and overall process, which solve the recalibration problem by fullfiling the above requirements.

3 Workflow recalibration

We propose three recalibration techniques. All techniques perform recalibration in an online manner, i.e. during workflow execution.

Recovery and monitoring points. This technique offers to the user manual recalibration. It enables the user to monitor intermediate results, make workflow changes and if the changes are in the already executed workflow part, then only the changed part is re-executed, avoiding to repeat computations; if changes affect only the non-executed part, then workflow changes are applied and execution continues.

Conditional points. This is an automatic technique that allows the execution of alternative predefined workflow branches.

Goto points. This is an automatic technique that conditionally changes an executed workflow part to a predefined alternative and re-executes it.

3.1 Recovery and monitoring points

This recalibration technique allows the user to change a workflow during its execution and avoids to unnecessarily repeat computations in the already executed workflow part. It involves the employment of two novel types of tasks: recovery and monitoring points. A *recovery point* rp_T is a task that stores the result of task T. A *monitoring point* mp_T is a task that invokes the visualization of the result of task T or part of it. We use the phrase *intermediate result* to refer to the result of a task T that has been executed, while the whole workflow execution is not yet finished. The visualization of intermediate results assists the user in making a recalibration decision.

Recalibration using this technique is performed in four steps: (1) the user augments a workflow with recovery and monitoring points and starts the workflow execution; (2) when the execution reaches a recovery point the system stores the intermediate results of the preceding task, required for a possible re-execution of the workflow part following this recovery point; (3) when the execution reaches a monitoring point the user observes intermediate results of the preceding task; the workflow keeps executing after the monitoring point, while the user observes the intermediate results; (4) the user changes the workflow part following a recovery point and performs a re-execution of the workflow from this recovery point and on.

When the user changes the workflow and re-executes it, PAW determines which intermediate results are required to re-execute the changed workflow part that follows a specific recovery point (or points). It prepares this workflow part as a new materialized workflow with these intermediate results as input datasets and sends it to IReS. The execution of the previous (original) workflow is aborted.

Figure 4 displays the workflow from the motivating example augmented with recovery and monitoring points. The user observes the result of *filter_region*s at *monitoring point*s, decides to change the parameters of *filter_region*s and re-executes the workflow from the *recovery point*s. So the most time-consuming part of *DataFilter* is not re-executed.

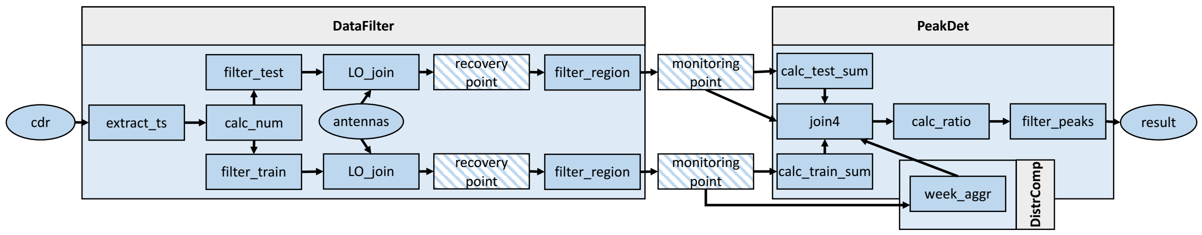

Figure 4: 'Peak Detection' workflow augmented with recovery and monitoring points

3.1.1 Implementations of points

Monitoring points invoke the visualization of the result of the preceding task. PAW includes monitoring points for specific operators, such as implementations of *k-means*, for which the result is visualized as a map of centroids or the histogram of cluster sizes. It also provides three basic monitoring operators, for the visualization of: geographical, numerical and categorical data. PAW includes recovery points for HDFS, Elasticsearch and operator-specific monitoring points for *k-means* and *tf-idf*.

3.2 Conditional points

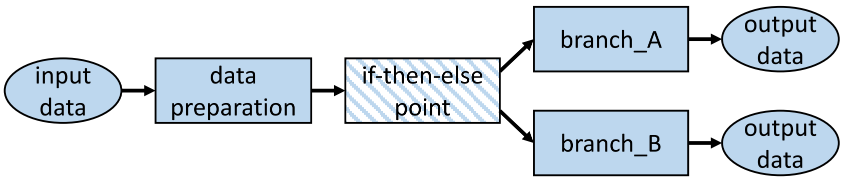

Figure 5: A workflow with an *if-then-else* point

PAW includes a new type of task that realizes the conditional structure of the form *if-then-else*. The latter allows the design of a workflow with several alternative workflow parts. Depending on the intermediate results of the task preceding the *if-then-else* task, a workflow branch is chosen for execution, over another one. These workflow branches are not yet executed. Figure 5 displays a workflow that has been augmented with one *if-then-else* point and two following workflow branches. The *if-then-else* task has two outputs; the boolean condition evaluates to true or false, depending on which PAW executes one of two branches.

The operator of the *if-then-else* point is implemented for any particular data. For example, for *tf-idf* PAW has an *if-then-else* task that evaluates if the weight of some word is above a certain value. Additional conditional points can be added through the interface of PAW.

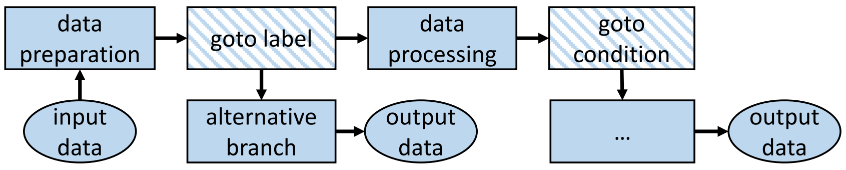

Figure 6: A workflow recalibrated with a *goto* point

3.3 Goto points

The workflow is augmented with two tasks: *goto label* and *goto condition* points, and an alternative workflow part related to the *goto label* point (Fig. 6). When the workflow execution reaches the *goto condition* point and if this task triggers 'goto' to *goto label*, then it re-executes the workflow from that point choosing for execution the alternative workflow part. Therefore, this technique is a combination of the *recovery and monitoring points* and *conditional* techniques.

The *goto condition* task has two outputs and a boolean condition evaluating to true or false, depending on which PAW continues execution or jumps to the *goto label* alternative workflow branch. The implementation of *goto condition* is similar to the *if-then-else* point.

3.4 Description of recalibration process

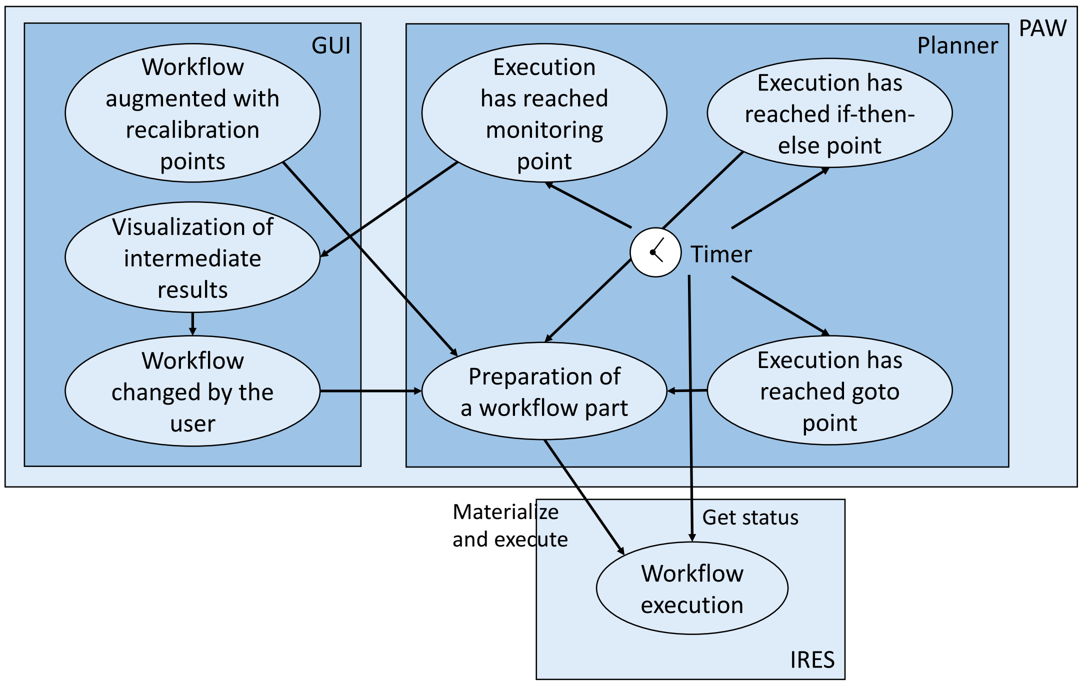
PAW
GUI
Workflow augmented with recalibration points
Visualization of intermediate results
Workflow changed by the user
Planner
Execution has reached monitoring point
Execution has reached if-then-else point
Timer
Preparation of a workflow part
Execution has reached goto point
Materialize and execute
Get status
Workflow execution
IRES

Figure 7: Recalibration process scheme

The above described techniques can be applied together to a single workflow. The process of recalibration unfolds as follows (Fig. 7):

1. The user augments a workflow with recovery and monitoring, *if-then-else* and *goto* points in GUI of PAW.
2. PAW prepares the workflow part 'till' the first *if-then-else* or *goto* point for execution. If there is no such points, it prepares for execution the entire workflow.
3. PAW sends the prepared workflow part to IReS for materialization and execution using REST API calls. More details about integration are in [4].
4. PAW periodically checks the status of workflow execution in IReS, which tasks have been executed and which are currently executing (*RunningWorkflows.getState()*).
5. When the execution reaches a monitoring point:
   (a) PAW displays the intermediate result in GUI to the user.
   (b) If the user changes the workflow, then PAW prepares a workflow part starting from the recovery point that includes the user changes.
6. When the execution reaches *if-then-else* or *goto* point PAW checks the result produced by this point and decides which workflow part has to be executed in the following.
7. If PAW prepared a new workflow part to be executed, then PAW aborts the previous execution and sends this new part to IReS.

4 Application of recalibration

In the following, we describe application scenarios of recalibration and demonstrate it on a set of workloads.

Workloads. We use synthetic and real workflows on real data. The real workflows and data come from the two use cases of ASAP [7] and belong to the domains of telecommunications and web analytics. One of the telecommunication workflows is described as a motivating example (Section 2.1). 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.

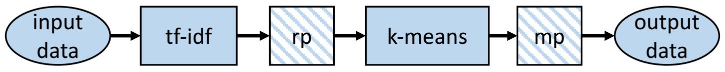

Figure 8: Application of *rp-mp* technique to k-Means workflow

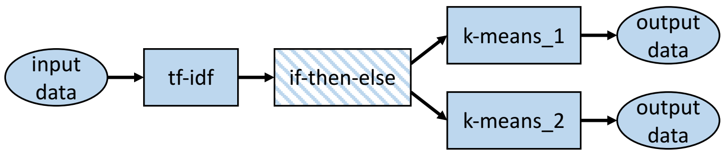

Figure 9: Application of *if-then-else* technique to k-Means workflow

Figures 8 and 9 display one k-means clustering workflow of a real web analytics use-case, on which we have applied two different recalibration techniques: recovery and monitoring points and conditional points, respectively. The workflow starts, as usual, with an input data extracted from the Web collections with an ElasticSearch query. Then *tf-idf* produces the term and inverse document frequencies. Further, *k-means* produces the clustering model and outputs the final result. The workflow displayed in Figure 8 is augmented with one recovery and one monitoring point. When execution reaches *mp* the user can see the cluster sizes in a form of histogram and can change the parameters of *k-means* and re-execute a workflow avoiding repeated computation of *tf-idf*. The workflow displayed in Figure 9 is augmented with an *if-then-else* point that compares the number of words, the score of which is above some prespecified threshold. *k-means_1* and *k-means_2* are the clustering tasks with a different number of centroids. Based on the result of *if-then-else*, either *k-means_1* or *k-means_2* is executed.

Figure 10 displays a workflow of an analysis of a product marketing campaign. It combines sales data with sentiments (*join1 by prod&reg*) about that product gleaned from tweets crawled from the Web (*calc sent&tag*). The result consists of total sales (*calc totalSales*) and average sentiment (*calc avgSent*) for each day of the campaign.

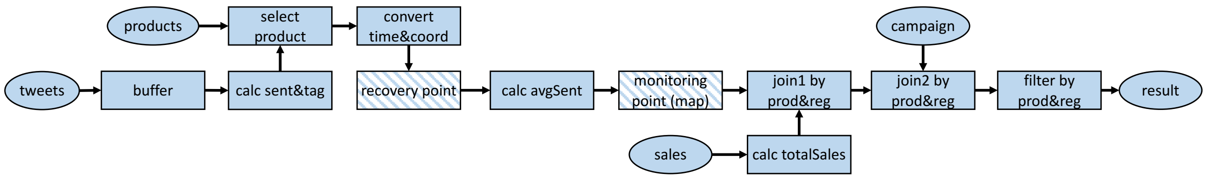

Figure 10: A workflow of marketing campaign analysis

Campaigns promote a specific product and are targeted at non-overlapping, geographical regions. To simplify the presentation, we assume the sentiment analysis of a tweet yields a single metric, i.e., like or dislike the product over a range of -5 to +5. In this use case the user places a monitoring point between *calc avgSent* and *join1 by prod&reg* that invokes the visualization of the geographical distribution of reviews. Observing this geographical distribution the user finds out that there is a peak of reviews in one region. Then he decides to limit the analyzed area to this specific region, (he adds a vertex *filter_region* right after *calc avgSent*) and changes the monitoring point to one that represents sentiment rating in a form of histogram. This histogram shows that there are a lot of negative reviews. Further, the user changes the workflow to show raw negative reviews in the analyzed area, by removing the part after the monitoring point, and finds out what is the reason of this "dissatisfaction peak". Figure 11 shows the workflow with all these changes made by the user. This use case demonstrates recovery and monitoring points technique with the infusion of a task.

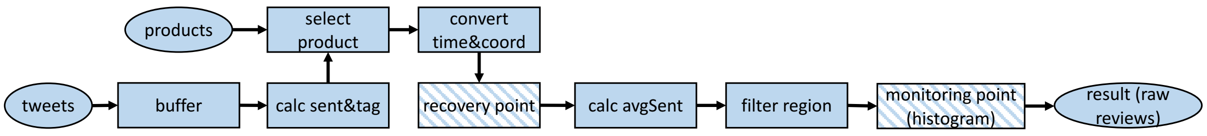

Figure 11: A workflow of marketing campaign analysis after two re-calibrations

Application scenarios of recalibration. The recalibration functionality of PAW can be demonstrated with four types of scenarios that aim to show each a distinct view of the recalibration benefits and its potential.

Scenarios A. These demonstrate the recovery and monitoring points technique. Specifically, they show real necessities to change workflows during execution. Above, we have described two real workflows which need infusion of new tasks or alteration of task parameters during the execution: Peak detection and k-means workflows.

Scenarios B. These also demonstrate the recovery and monitoring points technique. Specifically, they show how the user can design a workflow in a gradual and modular manner, while he is testing and debugging already created parts by monitoring intermediate results. These scenarios show how this workflow design process benefits exploratory data analysis. Above, we have described such a scenario on a synthetic workflow of marketing campaign analysis.

Scenarios C. These demonstrate the *conditional* technique for workflows with a natural conditional branching, for which data analysis based on some conditions follows different paths, and the selection between these alternative paths should be made at runtime. Above, we have described such a scenario on the k-means workflow and on one synthetic workflow.

Scenarios D. These demonstrate the *goto* technique using workflows that benefit from the *goto* point in order to find anomalies in data, narrow or refocus the search or analysis, as well as meet deadlines and milestones of analysis.

5 Related Work

Most Workflow Management Systems (WMS) are described in detail in the reports D5.1 [12], D5.2 [3] and D5.3 [13]. In this chapter we overview some of them and discuss works that engage in recalibration.

Multi engine systems Many systems have the ability to generate an execution plan from a logical. Here, Garlic [20], and TSIMMIS [24] were early prototypes of such systems. However, no systems are able to generate an execution plan for a variety of engines. Some systems can generate code within a particular family of engines. For example, some database query systems can generate SQL code for a variety of database systems (e.g., PostgreSQL, MariaDB [26]). Some systems have the ability to generate execution plans for more than one processing engine, for example BigDAWG Polystore System [21], which focuses on integrating independent storage engines. BigDAWG offers users *location transparency*, so that application programmers do not need to understand the details about the underlying database(s) that will execute their queries. This has been implemented using *islands of information*. Each island is a front-facing abstraction for the user, and it includes a query language, data model, and a set of connectors or shims for interacting with the underlying storage engines. Our system considers a wider configuration of a multi-engine platform; besides a set of datastores, there are many execution engines, such as a MapReduce engine or a scripting engine. Moreover, our model of logical workflow is more flexible and simple: to start using a new engine in BIGDAWG requires to define an *island* for it, in our system the user provides implementations in that engine of needed logical operators.

Many Extract-Transform-Load (ETL) systems (e.g., [15], [16]) can generate execution plans to, for example, filter and extract data from several database systems and then import and process that data into there native ETL engine. However, these systems are inflexible in that the data processing can be performed only on their internal ETL engine. Our system can generate a variety of execution plans from a single logical plan, e.g., given a logical plan, we might create one execution plan that filters a data set in a database system, while another plan might filter that data set using a scripting engine.

To the best of our knowledge, there is no previous work on a manual workflow recalibration at runtime. Also, existing workflow management systems do not provide recalibration based on *goto* methods. Some of them support conditional structures, but in a limited way: ASKALON is a grid application development and computing environment [28] based on AGWL [27], an XML-based workflow language, that supports *if* and *switch* structures. These are predefined for specific basic input and output.

Kepler [29] allows the design of scientific workflows and executes them efficiently using emerging Grid-based approaches to distributed computation. It offers several workflow control actors, that implements a conditional functionality on a high level: (1) a structure called *Comparator*, which takes two inputs and performs a numerical comparison ($<$, $<=$, $>$, $>=$, $==$), its output is a boolean result; (2) the *BooleanSwitch* actor has one data input and one control input. It has two output ports: *TrueOutput* and *FalseOutput*. Based on the value on the control input, data in the input port is forwarded to the output port; (3) the *Switch* actor has one data input port, one control input port and many output ports. Control input port selects one output port to forward the input data to that output port.

Kepler is derived from Ptolemy [30]. In Ptolemy, many actors have conditional behavior. For example, generic filters may use conditions to filter some tokens at the input ports to forward them to their output ports. Also, there exist workflow control actors, that implements a conditional functionality on a high level. The *Comparator* is a logic actor, which takes two inputs and compares them according to $<$, $<=$, $>$, $>=$, $==$. The output is a boolean result. The *BooleanSwitch* actor has one data input and one control input. It has two output ports: *TrueOutput* and *FalseOutput*. Based on the value on the control input, data in the input port is forwarded to the output port. Since Kepler does not have an *if* construct, *BooleanSwitch* actor can be considered as the closest construct to it. The *Switch* actor has one data input port, one control input port and a enumerated list of output ports. According to a number passed to control the input port this actor selects the corresponding output port to forward the input data to that output port.

Taverna [32] is a well-known workflow management system that does not include conditional structures in the workflow model, but tries to achieve the *if* and *switch* functionality at a higher layer of workflow management. Conditionals are driven by the control links within the workflow. As a process can only run when all upstream links, data and control, are satisfied it is possible to construct workflows where only a subset of the available downstream processors can run and where the others are left in the scheduled state. Effectively those chosen to run are in the 'true' branch of the conditional and those left scheduled in the 'false' branch. In Taverna such conditional behavior is implemented using processors *fail_if_false* and *fail_if_true* placed as first vertices of parallel branches. Depending of their input one of those processors fails, another satisfies and only satisfied branch continue execution.

UNICORE is a grid middleware, aims to provide seamless, secure and intuitive access to distributed resources [33]. UNICORE has a programming environment to design and execute workflows. It supports three specific types of *if-then-else* conditions, *ReturnCode*, *FileTest* and *TimeTest*. The first performs a numerical comparison, the second checks if a file exists or is executable and the third checks the current time.

Recalibration in PAW offers an abstract *if-then-else* task that can be customized for a variety of input data and complex conditions that involve the execution of fully-fledged procedures. Only Taverna offers same level of flexibility in the design of conditions as PAW does. The rest of the considered systems are very limited in possibilities to construct a condition.

6 Summary

This document describes novel techniques for recalibration of analytics workflows defined on multi-engine systems. This includes both automatic and manual techniques. Further, we showcase the application of the described recalibration techniques on a set of synthetic workflows and workflows of real-world applications.

References

[1] M. Filatov and V. Kantere. PAW: A Platform for Analytics Workflows. In *EDBT*, 2016.

[2] M. Filatov and V. Kantere. Multi-workflow optimization in PAW. (To Appear) In *EDBT*, 2017.

[3] V. Kantere, M. Filatov WP 5 - Adaptive Data Analytics. D5.2 Workflow management tool. Report, Aug. 2015.

[4] P. Pratikakis WP 7 - Integration of the ASAP System. D7.3 ASAP System Prototype. Report, Mar. 2017.

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

[6] V. Kantere and M. Filatov. Modelling processes of big data analytics. In *WISE*, 2015.

[7] Asap. `http://www.asap-fp7.eu/`.

[8] K. Doka, N. Papailiou, D. Tsoumakos et al. WP 3 - Intelligent, Multi-engine Resource Scheduling Platform. D3.2 IReS Platform v.1. Report, Aug. 2015.

[9] R. Bertoldi et al. WP 9 - Applications: Telecommunication Data Analytics. D9.2 Use Case Requirements. Report, Feb. 2015.

[10] P. Rigaux et al. WP 8 Applications: Web Content Analytics. D8.2 Use Case Requirements. Report, Feb. 2015.

[11] K. Doka, N. Papailiou, D. Tsoumakos and N. Koziris: IReS: Intelligent, Multi-Engine Resource Scheduler for Big Data Analytics Workflows. In SIGMOD, 2015.

[12] V. Kantere, M. Filatov WP 5 - Adaptive Data Analytics. D5.1 Workflow Management Model. Report, Feb. 2015.

[13] V. Kantere, M. Filatov WP 5 - Adaptive Data Analytics. D5.3 Data Processing Deployment. Report, Aug. 2016.

[14] K. Doka, M. Filatov, V. Kantere and N. Papailiou Optimizing, Planning and Executing Analytics Workflows over Multiple Engines. In *MEDAL*, 2016.

[15] Informatica ‘powercenter’. `http://www.informatica.com/products/powercenter/`.

[16] Oracle warehouse builder 10g. `http://www.oracle.com/technology/products/warehouse/`.

[17] Apache spark. `https://spark.apache.org/`.

[18] Apache hadoop. `http://hadoop.apache.org/`.

[19] Weka. `http://weka.pentaho.com/`.

[20] M. Tork Roth, M. Arya, L. Haas, M. Carey, W. Cody, R. Fagin, P. Schwarz, J. Thomas, and E. Wimmers. The garlic project. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD '96, pages 557–, New York, NY, USA, 1996. ACM.

[21] Aaron J. Elmore, Jennie Duggan, Mike Stonebraker, Magdalena Balazinska, Ugur Çetintemel, Vijay Gadepally, Jeffrey Heer, Bill Howe, Jeremy Kepner, Tim Kraska, Samuel Madden, David Maier, Timothy G. Mattson, S. Papadopoulos, Jeff Parkhurst, Nesime Tatbul, Manasi Vartak, and Stan Zdonik. A demonstration of the bigdawg polystore system. *PVLDB*, 8(12):1908–1911, 2015.

[22] A. Simitsis, K. Wilkinson, U. Dayal, and Meichun Hsu. Hfms: Managing the lifecycle and complexity of hybrid analytic data flows. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 1174–1185, April 2013.

[23] Alkis Simitsis, Kevin Wilkinson, Malu Castellanos, and Umeshwar Dayal. Optimizing analytic data flows for multiple execution engines. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 829–840, New York, NY, USA, 2012. ACM.

[24] Sudarshan S. Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *IPSJ*, pages 7–18, 1994.

[25] Christian Thomsen, Torben Bach Pedersen, and Wolfgang Lehner. Rite: Providing on-demand data for right-time data warehousing. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 456–465, 2008.

[26] Mariadb. `https://mariadb.org/`.

[27] T. Fahringer, J. Qin and S. Hainzer: Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language. In CCGrid, 2005.

[28] T. Fahringer, et al: ASKALON: A Grid Application Development and Computing Environment. 6th International Workshop on Grid Computing, 2005.

[29] B. Ludascher, et al: Scientific Workflow Management and KEPLER System, Concurrency and Computation: Practice & Experience. SI on Scientific Workflows, 2005.

[30] J. Buck, S. Ha, E. Lee and D. Messerschmitt. Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems. Int. Journal of Computer Simulation, 1994.

[31] L. Beltro Costa, L. Feitosa, E. Arajo and G. Mendes: MyGrid: A complete solution for running bag-of-tasks applications. In Proc. of the SBRC, 2004.

[32] Taverna. `http://www.taverna.org.uk/`.

[33] D.Erwin et al.: UNICORE Plus Final Report – Uniform Interface to Comp. Resources. The UNICORE Forum, 2003.

FP7 Project ASAP

Adaptable Scalable Analytics Platform

End of ASAP D5.4
Dynamic re-calibration of processing

WP 5 – Adaptive Data Analytics

Nature: Report

Dissemination: Public