Recalibration of Analytics Workflows

Maxim Filatov
Yandex, Moscow, Russia
maxfil@yandex-team.ru

Verena Kantere
School of Electrical and Computer Engineering
National Technical University of Athens
verena@dblab.ece.ntua.gr

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. Workflows become more complex and time-consuming to design and execute, since they span a variety of systems and the amount of data being processed grows. Therefore, it becomes increasingly difficult to debug workflows in order to handle errors, as well as adjust the workflow design and calibrate task parameters for applications that perform exploratory data analysis. Towards this end, the workflow management system should provide *recalibration* methods i.e. methods to monitor and to influence workflow processing at runtime. We demonstrate novel manual and automatic recalibration techniques for analytics workflows, 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. 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 while she creates the workflow 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 efficient error handling of complex and long-running workflows.

In this demonstration we focus on the novel functionality of PAW[1] (Platform for Analytics Workflows) 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] and includes the functionalities of workflow design and analysis in order to clarify execution semantics, single workflow optimization and multi-workflow optimization. In this demonstration 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).

Several existing workflow management systems support conditional structures to some extent. Each of these systems implements these structures in different ways and with some limitations: Kepler [7] allows the design of scientific workflows and executes them efficiently using emerging Grid-based approaches to distributed computation. It offers a structure called *Comparator*, which takes two inputs and performs a numerical comparison. Taverna [8] 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. 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 [9]. 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 the 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. Also, they don't provide *goto* statements.

A new system that offers interactive debugging framework for big data processing is BigDebug [10], which provides a set of debugging primitives: (a) simulated breakpoints and on-demand watchpoints that allow users to selectively examine intermediate data of computation; (b) data provenance capability, crash culprit determination, tracing and (c) capability to fix code at runtime by the user, avoiding a program re-run from scratch. Unlike PAW, BigDebug is a single-engine system and works only on top of Apache Spark. Moreover, debugging is only one of the several uses of the manual recalibration of PAW.

Overview of PAW. PAW implements a novel workflow model [4, 5]. A workflow W is a directed, acyclic graph (DAG) $G = (V, E)$. The vertices V represent data processing tasks T 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

[1] Source code and live demo can be found on https://github.com/project-asap/workflow

© 2018 Copyright held by the owner/author(s). Published in Proceedings of the 21st International Conference on Extending Database Technology (EDBT), March 26-29, 2018, ISBN 978-3-89318-078-3 on OpenProceedings.org.
Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

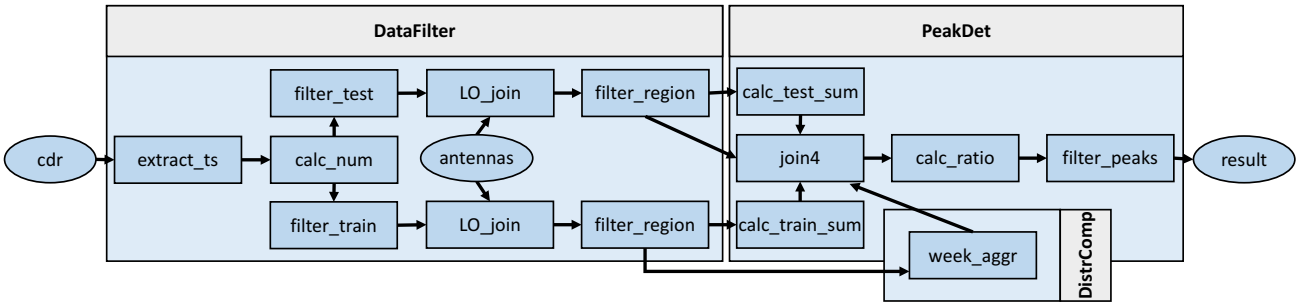

Figure 1: 'Peak Detection' of mobile calls workflow

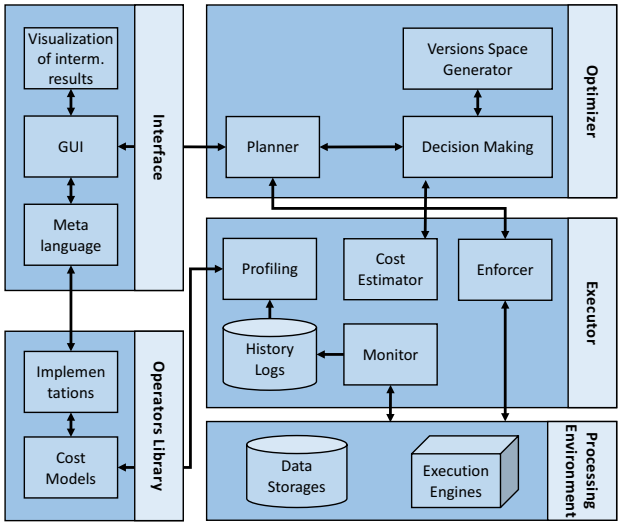

Figure 2: The architecture of PAW

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) [3], but it can also stand as an independent tool for workflow management and optimization. Figure 2 depicts the architecture of PAW, as well as its interaction with the rest of ASAP. ASAP components include execution, visualization of results, online adaptation etc. 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.

2 MOTIVATING EXAMPLE

Figure 1 shows a real-world analytics workflow from a telecommunication company, 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 and the hour of the day from the timestamp of each call record (*extract_ts*). The task *calc_num* counts the number of calls at one-hour intervals. Then, two filters split the data to training and test datasets. Further, the 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 the sum of calls in each day of the test and training datasets. Then, the test and training data are joined and the ratio of calls to the 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 on, for optimization needs [6], 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 changing each time the 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 she can observe faster call congestions and decide to change the search region. Furthermore, methods for automatic recalibration enable the expert to predefine conditions on intermediate results and, moreover, predefine decisions to be taken according to the outcome of condition evaluation.

3 WORKFLOW RECALIBRATION

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

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 re-execute only the changed part, avoiding to repeat computations; if changes affect only the non-executed part then workflow changes are applied and execution continues.

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

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

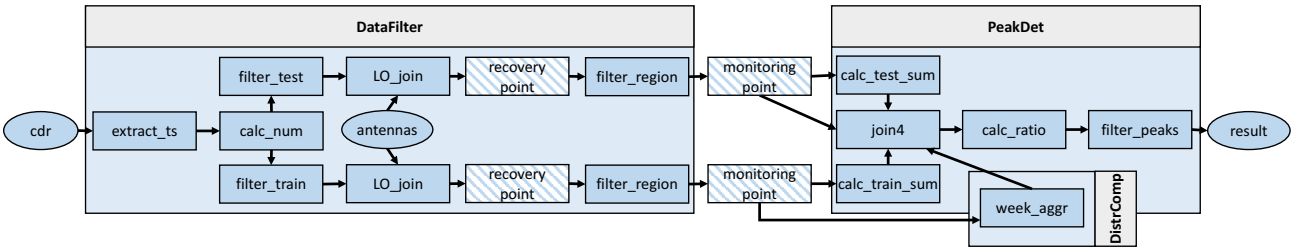

Figure 3: 'Peak Detection' workflow augmented with recovery and monitoring points

Figure 4: Monitoring intermediate results of 'Peak Detection' in PAW

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 runs it. The execution of the previous (original) workflow is aborted.

Figure 3 displays the workflow from the motivating example augmented with recovery and monitoring points. The user observes the result of *filter_regions* at *monitoring points*, decides to change the parameters of *filter_regions* and re-executes the workflow from the *recovery points*. So the most time-consuming part of *DataFilter* is not re-executed.

Figure 4 displays a 'Peak Detection' workflow in the interface of PAW. Green and yellow tasks are executed and currently executing, respectively. Using the geographical map on the bottom the user monitors intermediate results at the monitoring point marked with a blue stroke.

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 PostgreSQL and operator-specific monitoring points for *k-means* and *tf-idf*.

3.1.2 Partial execution. We propose two improvements of the *recovery and monitoring points* technique:

Execution of a task until a deadline or a milestone is reached. The preceding task of a monitoring point is executed for a predefined time (deadline) or until it has processed a predefined amount of the input data (milestone). The partial intermediate result of the task is received by the following monitoring point, which invokes its visualization. This partial intermediate result is observed by the user, who decides on recalibration faster.

Execution on a part of a dataset. Data processing between recovery and monitoring points is made on a part of the input dataset. This enables the user to observe intermediate results on a part of the input data, and take the recalibration decision faster.

There is no general method for preparing operators so that they produce partial intermediate results. However, in some cases we can have a general methodology. For example, we can break up a query into multiple queries by dividing the input data in chunks, and then combine the results after the execution of all statements. PAW has several SQL query operators, which can be split up in this way.

```
SELECT * FROM db
WHERE field BETWEEN 1 AND 10000;
SELECT * FROM db
WHERE field BETWEEN 10000 AND 20000;
...
```

3.2 Conditional points

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,

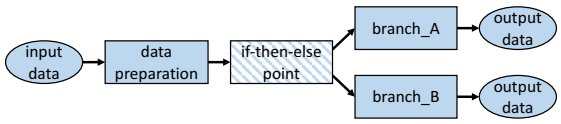

Figure 5: A workflow with an *if-then-else* point

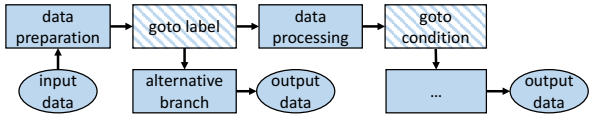

Figure 6: A workflow recalibrated with a *goto* point

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.

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.

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, four software platforms are running: Hadoop (CDH 4.6.0), Spark (1.4.1), Elasticsearch (5.1) and Weka (3.6.13).

Workloads. The demonstration uses synthetic and real workflows on real data. The real workflows and data come from the two use cases of ASAP [3] and belong to the domains of telecommunications and web analytics. One of the telecommunication workflows is described as a motivating example (Section 2). 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 recalibration functionality of PAW. It includes four types of scenarios that aim to show each a distinct view of the recalibration benefits and create discussion on the potential of recalibration of analytics workflows. 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 processing of the workflow, as well as review the internals of the platform, e.g. internal workflow representation. Even more, the participants are guided to play with the recalibration of workflows, by adding recovery and monitoring points, *goto* and *if-then-else* points, and change the workflow while monitoring intermediate results in GUI of PAW.

Scenarios A. These demonstrate the recovery and monitoring points technique. Specifically, they show real necessities to change workflows during execution. We show real workflows which need infusion of new tasks or alteration of task parameters during the execution.

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. We show how this workflow design process benefits exploratory data 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.

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.

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. In *EDBT*, 2017.
[3] Asap. http://www.asap-fp7.eu/.
[4] V. Kantere and M. Filatov. A framework for big data analytics. In *C3S2E*, 2015.
[5] V. Kantere and M. Filatov. Modelling processes of big data analytics. In *WISE*, 2015.
[6] K. Doka, M. Filatov, V. Kantere and N. Papailiou Optimizing, Planning and Executing Analytics Workflows over Multiple Engines. In *MEDAL*, 2016.
[7] B. Ludascher, et al: Scientific Workflow Management and KEPLER System, Concurrency and Computation: Practice & Experience. SI on Scientific Workflows, 2005.
[8] Taverna. http://www.taverna.org.uk/.
[9] D.Erwin et al.: UNICORE Plus Final Report – Uniform Interface to Comp. Resources. The UNICORE Forum, 2003.
[10] M. Gulzar, M. Interlandi, T. Condie and M. Kim: BigDebug: Interactive Debugger for Big Data Analytics in Apache Spark. in FSE, 2016.