FP7 Project ASAP
Adaptable Scalable Analytics Platform

ASAP D5.2
Workflow management tool

WP 5 – Adaptive Data Analytics

Nature: Report

Dissemination Level: Public

Version History

Version	Date	Author	Comments
1.0	18 Aug 2015	V. Kantere, M. Filatov	First Version
2.0	22 Aug 2015	V. Kantere, M. Filatov	Final Version
3.0	19 Apr 2016	V. Kantere, M. Filatov	Revised 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.

Abstract This deliverable is a report on the first version of the Workflow Management Tool (WMT). This version incorporates the prototypes of three core modules of the WMT architecture, namely the Workflow Design, Analysis and Optimization modules. The report first gives a quick overview of the WMT architecture and then delves into the implementation details of each involved module.

Keywords Workflow model, analysis, optimization, design tool.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Workflow management tool overview

The Workflow Management Tool (WMT) is a component of the ASAP system architecture. It is used for workflow creation, modification, analysis and optimisation.

WMT provides a GUI for workflow design. The model underlying WMT combines simplicity of expression of application logic and adaptation of the level of description of execution semantics. It enables the separation of task dependencies from task functionality. In this way, WMT can be easily, i.e. intuitively and in a straightforward manner, used by many types of users, with various levels of data management expertise and interest in the implementation.

The workflow and the including tasks are described using a JSON-based metadata language. The workflow is analysed and the result of the analysis can be a list of detected errors of a workflow or an analysed workflow which is actually an enhancement of the initial workflow with more vertices, substitution of vertices and/or edges in the initial workflow with others, and addition of metadata to the tasks.

WMT is a fully open-source[1] instrument that includes the designing interface, as well as analysis and optimisation modules.

1.2 Purpose of the document

This document serves as a report on the first version of the WMT and accompanies its prototype implementation. Its purpose is to delve into the implementation details of the core architectural modules. This includes the declaration semantics of the workflow, analysis and optimization techniques. Furthermore, we demonstrate the work of WMT on specific use-cases from D8.2 [13] and D9.2 [1]. Finally, we make an initial discussion on multi-workflow optimisation.

[1]`https://github.com/maxfil/wmt` - this will be changed to the ASAP repository

1.3 Document structure

The rest of this document is structured as follows:

- Chapter 2 gives a brief overview of the workflow model, including manipulation as described in D5.1 [19]. Moreover, it gives details on the workflow analysis and the workflow optimisation.
- Chapter 3 presents details of the current architecture and implementation of WMT modules.
- Chapter 4 guides the reader on how to use the tool from creation up to optimisation, describing the functionalities and then gives examples of workflows driven by the use-case scenarios of D8.2 [13] and D9.2 [1].
- Chapter 5 gives details on the current status of the formulation of some other optimisation objectives and multi-workflow optimisation and outlines our next steps.
- Chapter 6 summarises related work in the topic of the workflow optimization.
- Chapter 7 concludes the deliverable.

Chapter 2

Workflow management

This chapter gives an overview of the workflow model, its analysis and its optimisation, as it is defined in D5.1 [19], and gives details on what it is currently implemented in WMT. Moreover, it gives an update on our work in workflow optimisation.

2.1 Workflow model

The workflow represents applications as a directed acyclic graph (DAG) $G = (V, E)$. The vertices V in the graph represent application logic and the edges represent the flow of data. Application logic includes (a) the analysis of data, and (b) the modification of data. The edges E are directed and connect the vertices that produce and consume data. Each vertex in a workflow represents one or more tasks of data processing. Each task T is a set of *inputs*, *outputs* and a *processor*. In this text we also use the term *operator* instead of the term *processor*. Tasks may share or not inputs, but they do not share processors 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 flow of data.

Definition 1. *In a digraph G, the out-degree $d^+_G(v)$ is the number of edges leaving a vertex v, and the in-degree $d^-_G(v)$ is the number of edges entering a vertex v.*

Definition 2. *A root is a vertex v with in-degree $d^-_G(v) = 0$; and a sink is a vertex with out-degree $d^+_G(v) = 0$.*

A workflow has one root and several sink vertices. 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 her workflow whereas materialized are the actual operator implementations and existing datasets, either provided by the user or residing in a repository. A generic tree-metadata format depicted as a subtree with a root *'operator'* is shown in Figure 2.1. In Section 3.1.1 (Tree-metadata framework) of the report D3.2 [8] meta-data of data and operators are described in more detail.

2.1.1 Workflow representation

In WMT a workflow is represented in a single JSON file. This representation captures structural information, design metadata (e.g., functional and non-functional requirements, physical characteristics like resource allocation), operator properties (e.g., type, schemata, statistics, engine and implementation details, physical characteristics like memory budget), and so on.

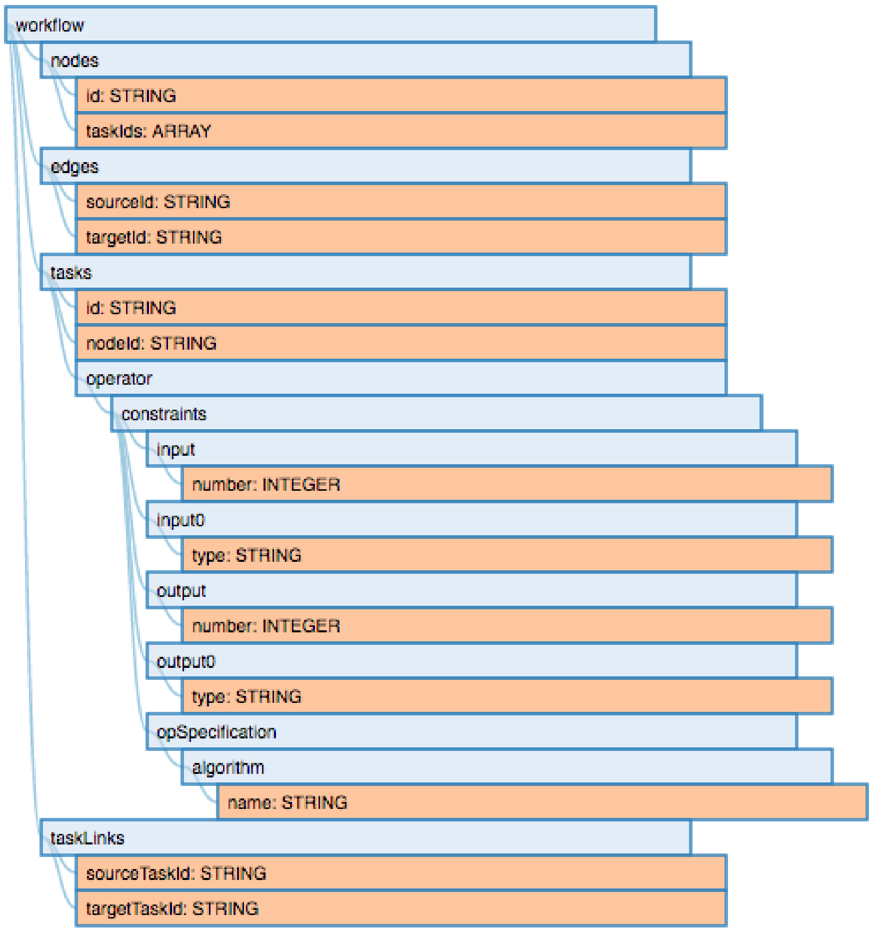


Figure 2.1: Workflow metadata tree

The first levels of the metadata tree of the workflow are the following (shown in Figure 2.1):

- **Nodes** Each node contains a list of task IDs which belong to this node.

- **Edges** This is a list of pairs of node IDs - $(sourceId, targetId)$. An edge defines the flow of data from one vertex to another. These nodes are called the source and the target, respectively.

- **Tasks** This part contains a list of task meta-data. The task meta-data consists of the information that is used to match abstract and materialized operators and datasets and the ID of a node to which this task belongs.

- **TaskLinks** (optional) This part contains links between tasks lying within a single node.

2.2 Workflow analysis

The workflow structure alleviates from the user the burden of determining any or some execution semantics for 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.

More specifically, in the analysed workflow, an edge with different input and output metadata, may be replaced with two edges and a new vertex; the new vertex corresponds to a new task that takes the data and metadata of the input of the initial edge and produces the data and metadata of the output of the initial edge. In other words, since the data of the input and the output of an edge are equivalent, this task changes only the metadata. Such vertices are *associative*, as they encompass associative tasks. Also, a vertex that includes multiple tasks, in the original workflow, is replaced, in the analysed workflow, with a set of new vertices that each includes one task of the original vertex. The new vertices may or may not be connected with new edges.

Furthermore, in the analysed workflow, 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. This set contains vertices that correspond to the tasks of the initial vertex: each new vertex corresponds to one task; vertices may correspond 1-1 to tasks, but it can be the case that two or more vertices correspond to the same task[1]. Naturally, the incoming edges of the initial vertex may have to be replicated, since they may correspond to the input of more than one tasks. The outgoing edges, however, remain the same, as each corresponds to the output of one task. The replacing subgraph may also contain new edges that connect the replacing vertices. Such edges represent the dependencies between tasks related to their

[1]Replication of tasks using many associative vertices that correspond to the same task of an original vertex may be necessary for the optimisation of the workflow execution.

execution semantics, and not related to the semantics of the application logic, as expressed by the user.

2.3 Workflow optimisation

In D5.1 [19] we have proposed two axes of a workflow optimization, namely: optimisation via graph reconfiguration and via optimal resource management. Currently, we focus on the first axis of optimisation, i.e. via the reconfiguration of the graph of the analysed workflow. To do this we employ methods for the manipulation of the workflow. A brief overview of these methods is provided below.

2.3.1 Workflow manipulation

A workflow is manipulated so that it can be executed more efficiently than originally designed. Manipulation is performed using the following operations:

- **Swap.** The *swap* operation applies to a pair of vertices, $v1$ and $v2$, which occur in adjacent position in an workflow graph G, and produces a new workflow graph G' in which the positions of $v1$ and $v2$ have been interchanged. The goal of the *swap* operation is to change the execution order of tasks. Currently, the WMT uses the *swap* operation in the optimisation module.
- **Merge.** The *merge* operation takes as an input two vertices and produces one new vertex that includes the tasks of both initial vertices. The vertices that are merged can be connected with an edge, i.e. together they represent some task dependency(ies), or not, i.e. there is no task dependency between them. The goal of the *merge* operation is to allow for a united optimisation of the tasks included in the two initial vertices, e.g. joint micro-optimisation on an execution engine. Currently, this operation is not yet implemented in the WMT.
- **Split.** The *split* operation takes as input one initial vertex and produces two new vertices that, together, include all the tasks included in the initial vertex. The two new vertices may or may not be connected. The goal of the *split* operation is to lead to separate optimisation of subgroups of tasks included in the initial vertex. Currently, this operation is implemented in WMT and used to produce the *analysed* workflow.

2.3.2 Operator characteristics

Workflow manipulation can be performed selectively depending on operator characteristics:

- **Blocking operators** require knowledge of the whole dataset, e.g., a grouping operator or an operator *join* or *sort*.

- **Non-blocking operators** that process each tuple separately, e.g., operators *filter* or *calc*.
- **Restrictive operators** output a smaller data volume than the incoming data volume, e.g. *filter*.

Table 2.1 shows the operators that are currently in the operator library in WMT and their categorization.

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 2.1: Operator categorization

The *filter* operator returns all rows for which the <filter_predicate> is 'True'. The *calc* operator produces data with new attribute <calc_attr> and value calculated by <calc_expression>. The others are obtained from the IReS platform.

For any operator that is added in WMT, the user has to define combinations of the new operator and operators already in the library, on which the *swap* operation can be applied, if there are any such combinations. The WMT can apply swapping on these predetermined combinations during the optimization stage. For example, the *groupBy Sort/filter* combination is always swappable, while the *calc/filter* combination is swappable or not depending on the specific implementation of *calc*.

2.3.3 Single-workflow optimization

We formulate the problem of optimizing a workflow for a single engine as a state space search problem. Starting with an initial workflow graph, we apply a set of graph transitions (see Section 2.3.1) to create new, equivalent graphs with (possibly) different costs. Applying transitions creates a large state space and the goal is to find an optimal workflow in this space with respect to the objective function (see Chapter 5).

We explore the state space exhaustively using the Exhaustive Search (ES). ES generates all possible states by applying all the applicable transitions to every state. The vastness of the state space requires more efficient exploration methods than the Exhaustive Search. To improve the search performance, a solution is to prune the state space. In the following, we propose techniques for achieving this.

Heuristics that can drive us close to the optimal solution quickly are the following:

- **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, e.g., rather than *filter →sort →function →group* do *filter →function →sort →group.*
- **H**3: Parallelize non-blocking operators. Place adjacent operators on parallel paths, so that the latter can be executed on separate physical processors, e.g., if there are adjacent operators *filter1 →filter2*, create two new paths in the workflow and assign the *filter1* and *filter2* operators each to a different path, so that these workflow paths can run on separate processors concurrently. The workflow paths or, in general, parts, to be executed in parallel should be chosen such that their latency is approximately equal, i.e., if parallelization is possible on n processors, break the flow into n workflow parts $x_1, \ldots, x_n$ of equal (or near-equal) execution time, such that $max(time(x_i))$ is minimized.

Currently, the workflow optimization in WMT is carried out using the operation *swap* and following the heuristics $H1$ and $H2$ (for the current form of the optimization algorithm see Section 3.3).

Chapter 3

Workflow management tool architecture and implementation

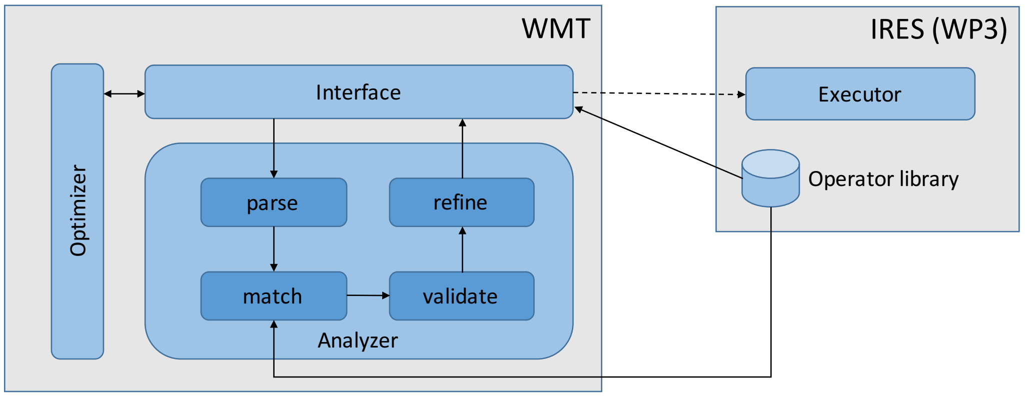


Figure 3.1: The architecture of the WMT

In this chapter, we describe in detail the current implementation of WMT. We discuss the architecture, the functionalities provided by the different modules and integration of WMT with other parts of ASAP. Figure 3.1 depicts the architecture of WMT as well as its interaction with external components. The main components of the architecture are:

- **Interface.** The interface accepts a workflow definition in the representation described in Section 2.1.1. It 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 a library of operators supported in WMT (See Section 3.4), 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 graph optimized towards the performance objective (for more on the performance objective see Section 5.1).

These provide for workflow design, analysis and optimization. After their design, analysis and optimization, workflows are ready for their execution. They can be executed on independent engines and storage repositories, i.e. engines and repositories that may be accessed through other paths in ASAP (Section 3.4) or another third party platform, besides WMT.

3.1 WMT architecture

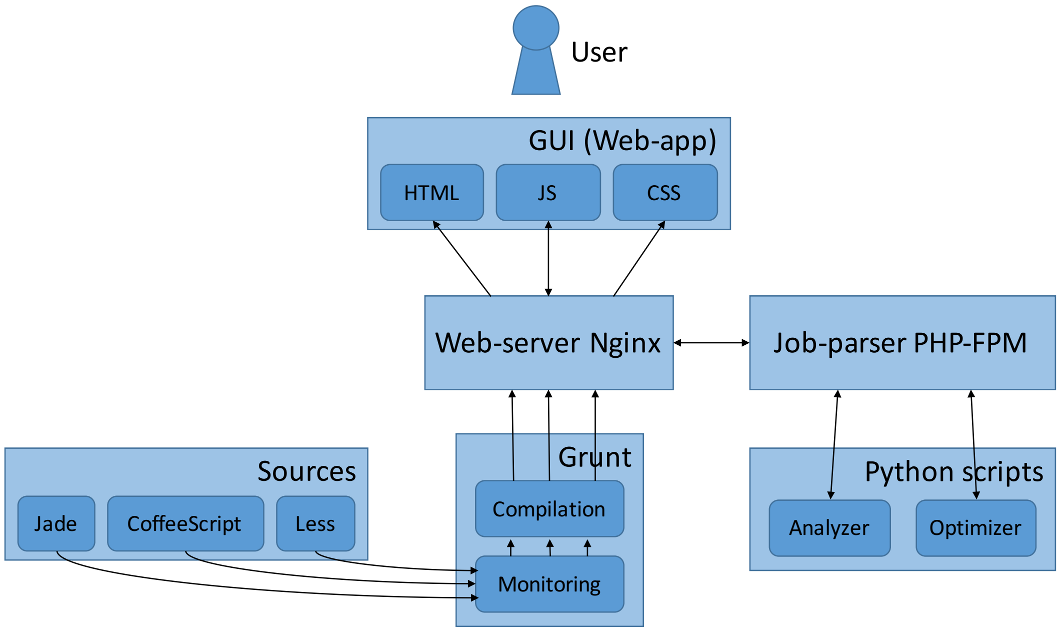


Figure 3.2: Technology stack used in WMT

WMT interface is a web application. It provides full functionality for designing a workflow even in the absence of server-side (Analyzer and Optimizer modules). It is encoded in Hypertext Markup Language (HTML [4]). To deliver content, WMT uses the Nginx [10] web server. To encode business logic, WMT uses Javascript [7] and PHP-FPM [11]. The pages and scripts are compiled from Jade [6] and CoffeeScript [2] sources, respectively, using Grunt [3]. The Analysis and Optimisation modules are scripts in Python [12]. Figure 3.2 depicts the technical stack used in WMT and the interaction of WMT parts.

3.2 Analyzer

WMT analyses a workflow in several steps:

1. Parsing the workflow.
2. Categorizing operators (see Section 2.3.2).
3. Validating consistency. A workflow is checked for the existence of cycles (Tarjan's Algorithm [18]) and correspondence of metadata of adjacent nodes. Cycle discovery and metadata mismatch of adjacent nodes fall into the error list. The errors with the cycles cannot be resolved and the analysis stops and returns a list of errors back to the Interface. If possible, the data flow errors are solved by adding associative tasks in Step 6.
4. Generating metadata of edges. These are a joint result of input and output metadata of source and target nodes, respectively.
5. Splitting several tasks in a single node to several single-task nodes.
6. Augmenting the workflow with associative tasks. Currently, the implemented tasks are converting data flow: buffer and format conversion.

3.3 Optimizer

The Optimizer works on the analyzed workflow, where tasks have been categorized and each node has a single task. Currently, the Optimizer uses heuristics $H1$ and $H2$ described in Section 2.3.3. These move restrictive operators towards the root of the workflow and place non-blocking operators together and separately from blocking operators.

The current form of the optimization algorithm follows all paths in a workflow by encountering all edges. It swaps nodes following the heuristics. It stops when no more swaps can be performed and an optimized version is produced (See Algorithm 3.1).

Algorithm 3.1: Optimization algorithm

```
Input: An initial workflow $W$
Output: An optimized by two heuristics equivalent workflow state $W^o$
$W = (nodes, edges, tasks)$
while $swaps <> 0$ do
  $swaps = 0$
  for edge in edges do
    $source = nodes[edge[sourceId]]$
    $target = nodes[edge[targetId]]$
    if $d_G^+(source) <> 1$ or $d_G^-(source) <> 1$ then
      continue
    if $d_G^+(target) <> 1$ or $d_G^-(target) <> 1$ then
      continue
    # heuristic 1
    if $category(task(source))$ is $`restrictive'$ then
      continue
    if $category(task(target))$ is $`restrictive'$ then
      $swap(source, target)$
      $swaps++$
      continue
    # heuristic 2
    if $category(task(source))$ is $`non-blocking'$ then
      continue
    if $category(task(target))$ is $`non-blocking'$ then
      $swap(source, target)$
      $swaps++$
      continue
$W^o = (nodes, edges, tasks)$
return $W^o$ end
```

The employed heuristics are based on the workflow topology as well as on the categorization of operators. In the future, the Optimizer will also employ estimations about the processing cost of workflows, which will be derived from the IReS Model db.

3.4 Integration

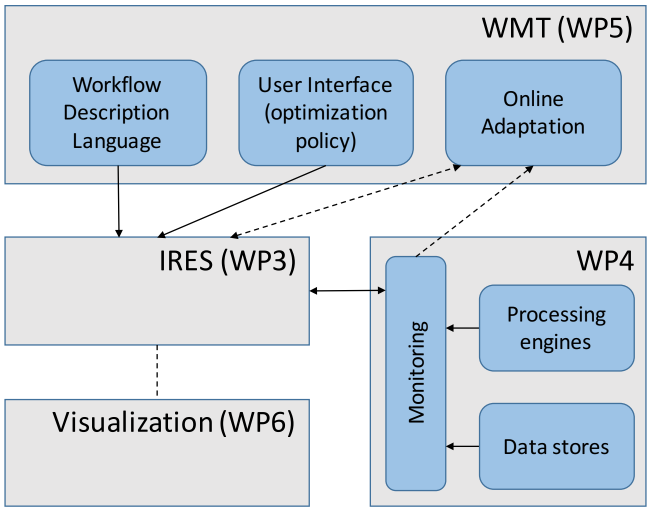


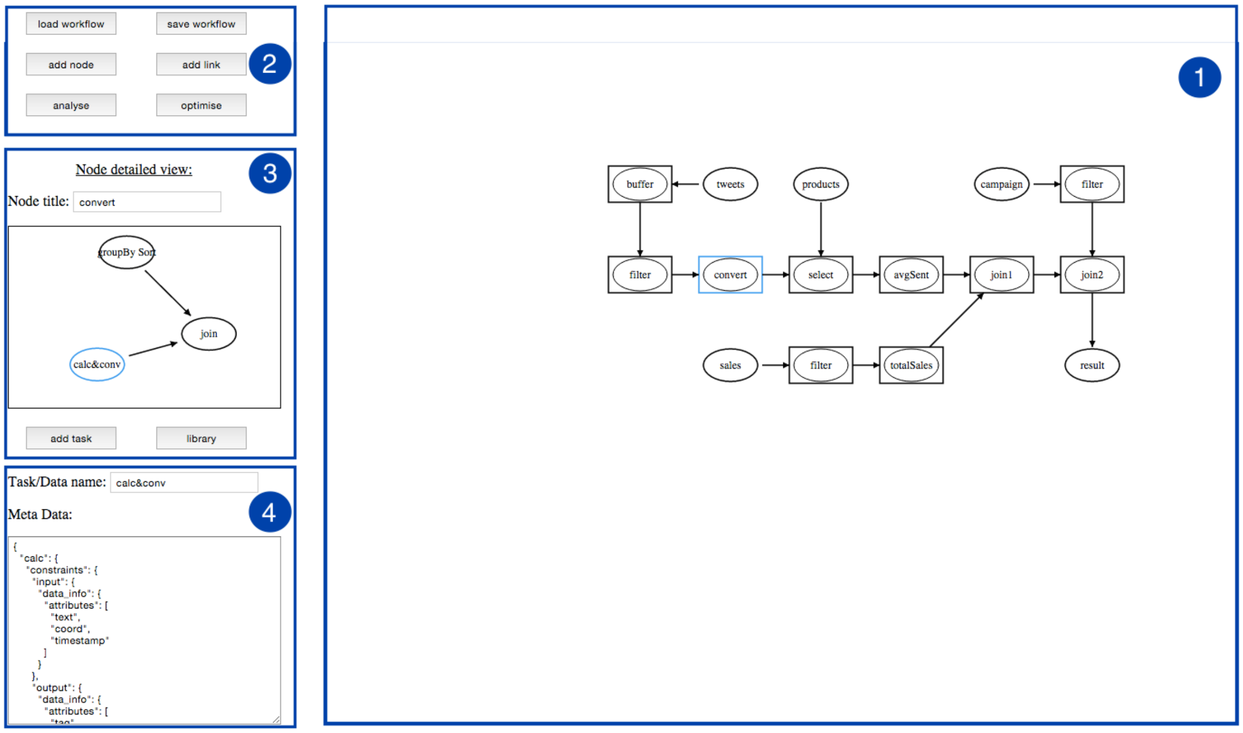
Figure 3.3: WMT interaction with the rest components of ASAP

IReS is a component of the ASAP that executes workflows (See D3.2 [8]), therefore WMT will have a tight integration with it (Figure 3.3). Currently WMT uses the Tree-metadata language (See 3.1.1 in the document D3.2 [8]) to express operators and receives a list of existing operators from IReS (See 2.3 in the document D3.2 [8]).

Chapter 4

Workflow management tool functionality

4.1 GUI



Node title: convert

Task/Data name: calc&conv

Meta Data:

```
{
  "calc": {
    "constraints": {
      "input": {
        "data_info": {
          "attributes": [
            "text",
            "coord",
            "timestamp"
          ]
        }
      },
      "output": {
        "data_info": {
          "attributes": [
            "tag"
```

Figure 4.1: GUI of WMT

WMT provides a GUI to enable users to design workflows and perform analysis and

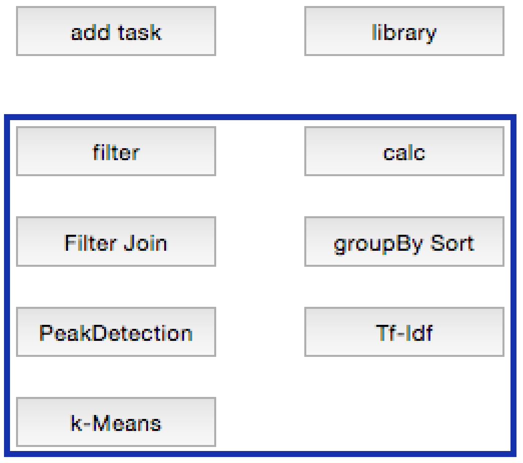


Figure 4.2: Library of abstract operators

optimization. The GUI consists of several areas (Figure 4.1) that perform the following functions:

- Display the workflow (Area 1).
- Add nodes and edges (Area 2). This process is depicted in Figure 4.3. First, the user adds a node, then she adds two tasks in it from the operators library, and finally she connects nodes and tasks.
- Create a new workflow from scratch, save and load it.
- Perform workflow analysis or optimization.
- Add tasks from a library (see Figure 4.2) or create new ones (Area 3). If the user adds a task from a library then it is accompanied by a set of metadata, i.e., properties that describe them. If a new task is created then the first levels of the metadata tree are predefined but users can add their ad-hoc subtrees to define their custom data or operators.
- Display metadata of the selected task (Area 4).

4.2 Examples

The ASAP focuses on the real-time analysis of Web content and telecommunications data. This section presents several indicative use-cases selected based on their relevance to the ASAP research.

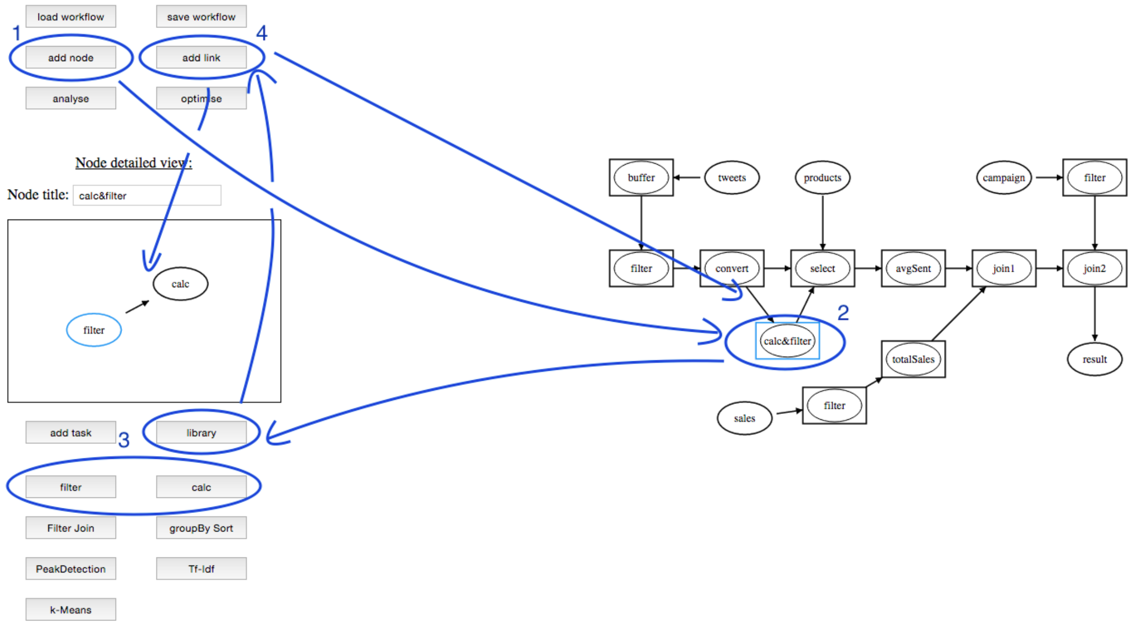


Figure 4.3: Steps of adding node

4.2.1 Web content analytics

The use case in this domain are centered on the services by Internet Memory Research as part of the Mignify platform (`www.mignify.com`). These services provide access to a very large collection of contents extracted from the Web, cleaned, annotated and indexed in a distributed infrastructure mainly based on Hadoop components. ASAP focuses on extending and enriching the public workflow interface supplied by Mignify, referred to as pipes (queries associated with a set of intelligent agents to extract or transform large-scale web data).

Stream processing

This use case captures a typical form of the current IMR Web analytics pipeline. Figure 4.4 presents a workflow for this use case. Data are selected from the document store based on some conditions (*select*). These data are processed in order to extract some text (*calc*), and the extracted text is moved to a different data store and stored there with other text and annotations (*move*). The output data are further processed via NLP-classification.

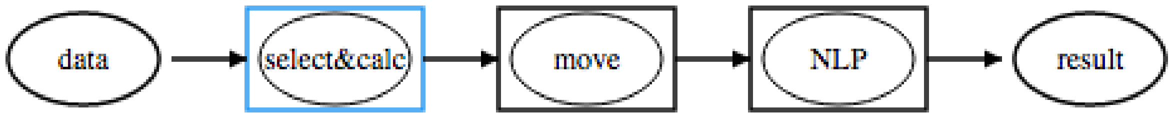


Figure 4.4: Workflow for NLP-classification

Figure 4.5 shows the result of the analysis module. The node containing two tasks select and calc has been split into two single-task nodes.

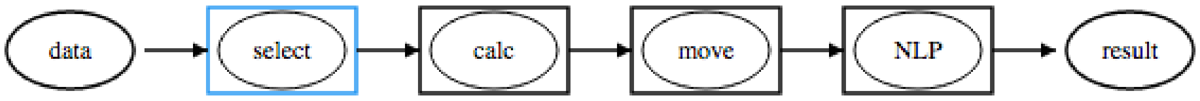


Figure 4.5: Workflow for NLP-classification

4.2.2 Telecommunications data

Call Detail Records (CDR) data is a good proxy to understand human mobility. The sheer volume of this data poses new challenges when extracting and visualizing specific indicators. ASAP investigates applications such as the following:

Peak detection

This use case involves processing of the anonymised CDR data of the past day by first selecting a spatial region and a temporal period (*select*). For this region and period, the number of calls is calculated (*calc*). Data and calculations from CDR are archived (*archive*) in other storage (*history*). After calls are counted, the application proceeds with algorithmic processing that detects peaks (*calc2*). The objective of this processing is to detect peaks in load, according to a set of criteria. Criteria may include the minimum size of a region and/or period, the cut-off distance, or other parameters for selecting regions and periods. These parameters should be adjustable by the analytics engineer, marketing expert, etc., who uses the peak analysis results. The results of this workflow are added to a database (relational or graph DBMS) that contains peaks detected in previous data. The database of peaks can then be queried by a user to discover clusters of calls that occur with regularity e.g., every week, discover clusters of calls that occur without any regularity, or similar ad-hoc queries based on the pre-computed peak data. The workflow for this use case is shown in Figure 4.6.

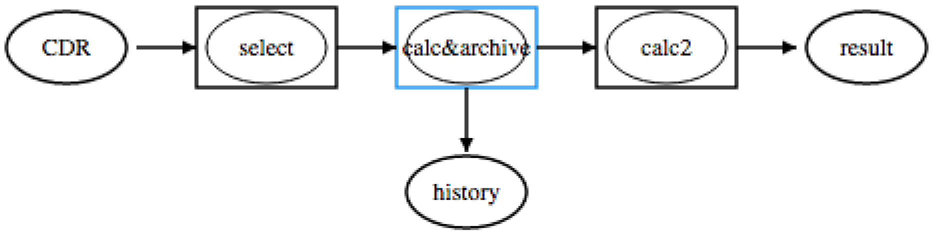


Figure 4.6: Workflow for the detection of peaks

Figure 4.7 shows the result of the analysis module. The node containing two tasks calc and archive has been split into two single-task nodes.

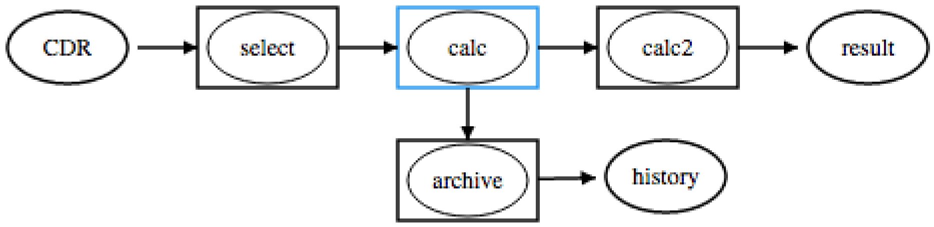


Figure 4.7: Analysed workflow for the detection of peaks

4.2.3 Marketing analytics

Figure 4.8 displays workflow of an analysis of a product marketing campaign. It combines sales data with sentiments about that product gleaned from tweets crawled from the Web. The result consists of total sales and average sentiment for each day of the campaign. 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.

Figure 4.9 represents an analysed workflow. The Analyzer splits two multi-task vertices (*convert time&coord*, *calc avgSent* and *filter by prod&reg*, *join2 by prod&reg*) and adds an associative tasks *buffer*. It is added to convert streaming data to batch.

Figure 4.10 depicts the result of the Optimizer work. It makes several workflow graph manipulations: (a) the vertex *filter by prod&reg* is factorized[1] and pushes closer to data-sources (sales, campaign, tweets); (b) The vertex *convert time&coord* is swapped with *select product* and merged with *calc sent&tag*, producing the vertex *calc&conv*. We have presented the example of marketing analytics, as described here, in [9].

[1]This and two more transitions are part of future work and will be described in the next deliverable.

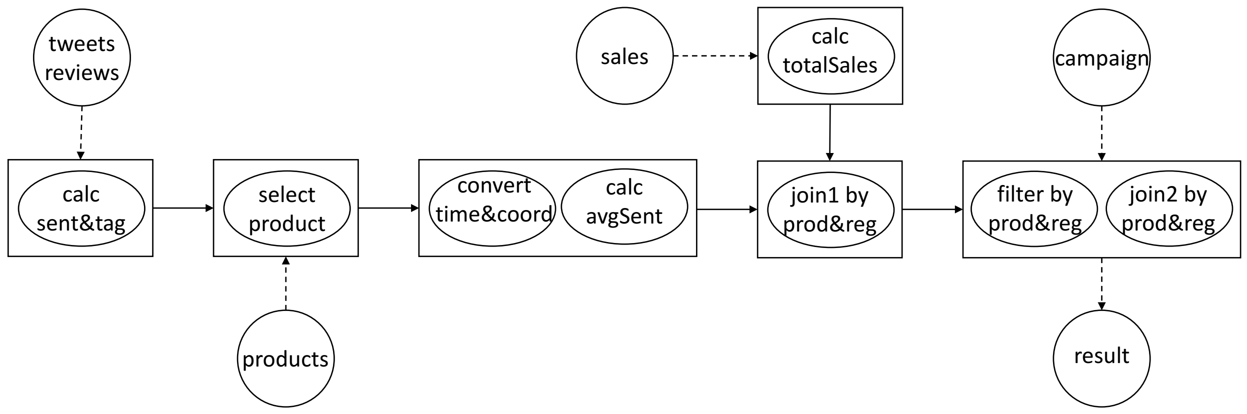


Figure 4.8: Workflow analysing a product in a marketing campaign

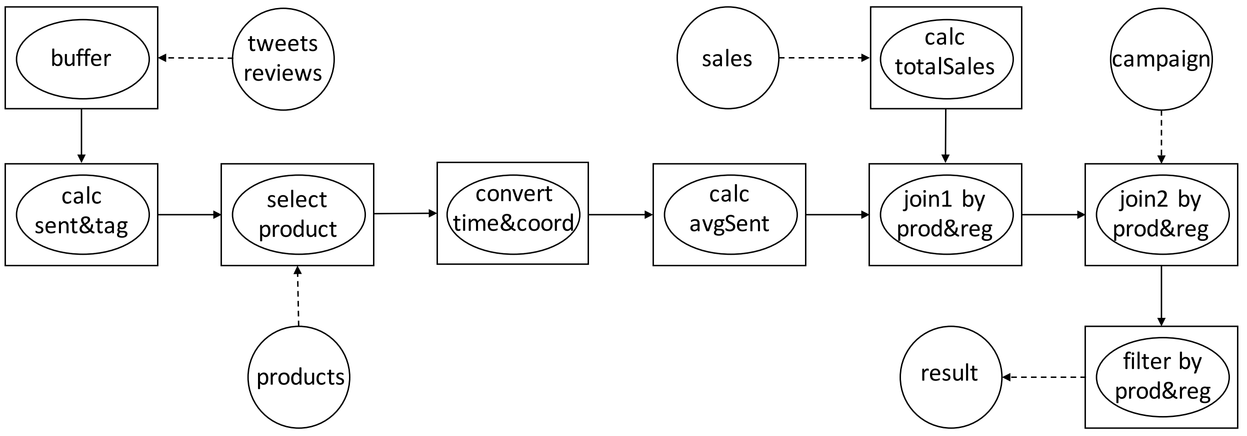


Figure 4.9: Analysed version of the workflow in Fig. 4.8

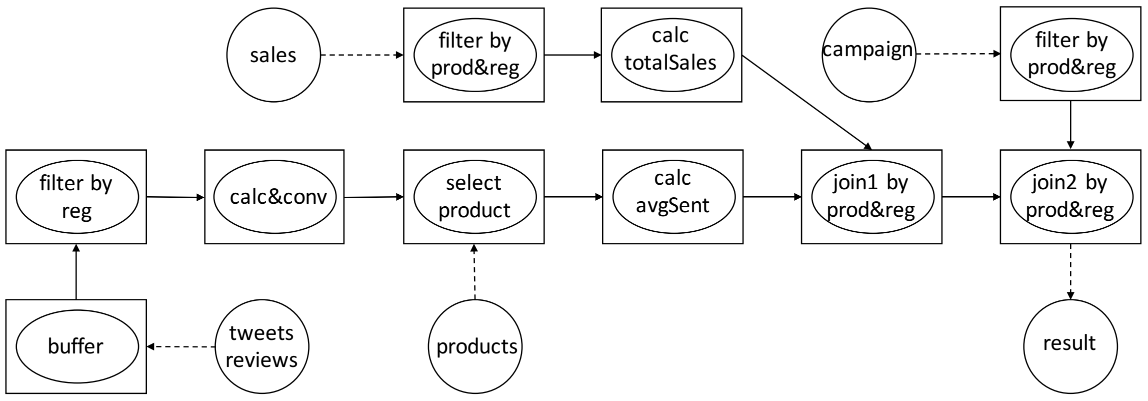


Figure 4.10: Optimised version of the workflow in Fig. 4.9

Chapter 5

Ongoing and future work

This chapter presents ideas that are towards a multi-workflow optimization.

5.1 Performance objective

Performance is a metric of workflow execution expressed in terms of time and used resources.

Let $W = \{W_1, \dots W_n\}$ be a list of workflows. These workflows arrive at time points $T^a = \{t^a_1, \dots t^a_n\}$. These two lists are not fixed, but changing over time as new workflows are added. The sizes of lists are equal and initially the list of times consist of zeros and $\forall i < j, t^a_i \leq t^a_j$. Workflow $W_{(n+1)}$ arrives at $t^a_{(n+1)}$ and these elements append to the lists and so on.

A workflow W_i is a graph G as defined by the workflow model. Each manipulation operation (see Section 2.3.1) transforms a workflow graph G into an equivalent graph G'. In other words, if an operation f is applied on a workflow W_i it produces a new workflow W'_i: $W'_i = f(W_i)$. Two workflows are equivalent if they produce the same output, given the same input.

All possible equivalent workflows, or else, workflow states, can be produced by applying all the applicable operations to every equivalent workflow state. We denote as $SL(W_i) = \{W^1_i, \dots W^k_i\}$ the list of all equivalent workflow states. $T^s = \{t^s_1, \dots t^s_n\}$ and $T^e = \{t^e_1, \dots t^e_n\}$ are the lists of start and end times of execution for workflows respectively.

The goal is to minimize the total elapsed time of completion of the execution of workflows:

$$\min \sum_{i=1}^{n} (t^e_i - t^a_i)$$

The end time of execution t^e of a workflow depends on the starting time of execution t^s, as well as the *processing cost* C of this workflow. The cost C is related to the time intervals the workflow is processed on one or several machines (i.e. a parallelized execution) and depends on the state of the system, (i.e. the utlization of resources by a pool of workflows), at a specific point in time. Consequently, the objective function is:

$$OF(t) = \min \sum_{i=1}^{n(t)} (t_i^s - t_i^a + C(W_i, t))$$

The estimations of the processing cost of workflows will be an input from the IReS platform.

5.2 Multi-workflow optimization

Beyond optimising single workflows, we will explore the optimisation of multiple workflows. We will aim to find common or similar subgraphs that we can optimise once and execute once or a few times.

Definition 3. *A 'common part' is a subgraph that is part of two or more workflows; these parts are bijective and corresponding operators in vertices are equivalent.*

Thus, our approach for multi-workflow optimization will be the joint execution of common parts (Definition 3). We will try to rearrange workflows with common parts in order to enable such execution (see Figure 5.1).

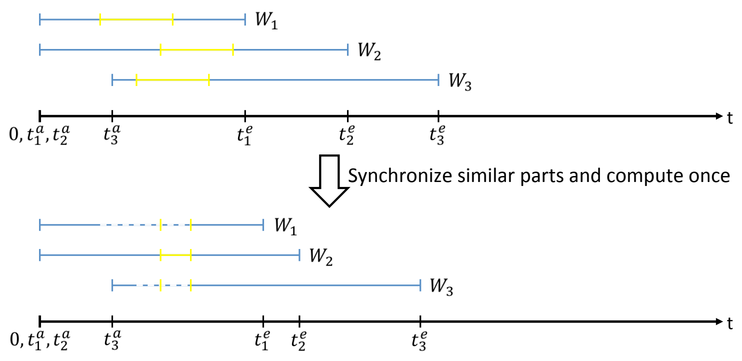


Figure 5.1: Synchronisation of common parts of multiple workflows

However, there are cases of workflows with common parts for which such rearrangement, and, therefore, joint execution of common parts, is not possible. Such cases occur if subgraphs of common parts have dependencies on subgraphs of other common parts. We introduce the terminology of 'mutual arrangement' of subgraphs:

Definition 4. *A vertex v is reachable from another vertex u, if there is a directed path that starts from u and ends at v.*

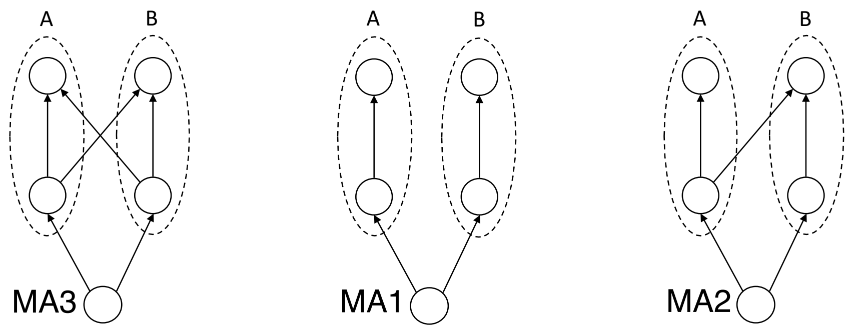


Figure 5.2: Mutual arrangement of subgraphs A and B

Definition 5. *A subgraph S depends from a vertex v, if there exists a vertex u in the subgraph and u reachable from v.*

Definition 6. *The mutual arrangement of two subgraphs A and B may be (Figure 5.2):*

- ***MA1:*** *A and B are independent, if there does not exist a pair of vertices of the two subgraphs, for which one vertex depends on another.*
- ***MA2:*** *Uni-dependent (A depends on B), if there exists a vertex v in B and A depends on v, but there does not exist a vertex in A on which B depends.*
- ***MA3:*** *Cross-dependent (A and B are cross-dependent if there exist vertices v in A, u in B and B depends on v, A depends on u.*

Depending on the mutual arrangement of subgraphs that belong to common parts of workflows (within or across workflows), it may or not be possible to perform joint execution of more than one common parts. We will showcase this with an example. Let us consider the two workflows W_1 and W_2 that have two common parts, A and B. Joint execution of both A and B is not possible if:

- At least in one workflow the subgraphs of common parts A and B are cross-dependent (MA3).
- In workflow W_1 common part A depends on common part B (MA2) and in W_2 B depends on A (MA2), as depicted in Figure 5.3.

We will devise techniques in order to discover efficient dependencies of common parts. Towards this goal we will be inspired by the problem and solutions of the Longest and Heaviest Increasing Subsequence [5, 20], which seem very promising for dealing with such situations.

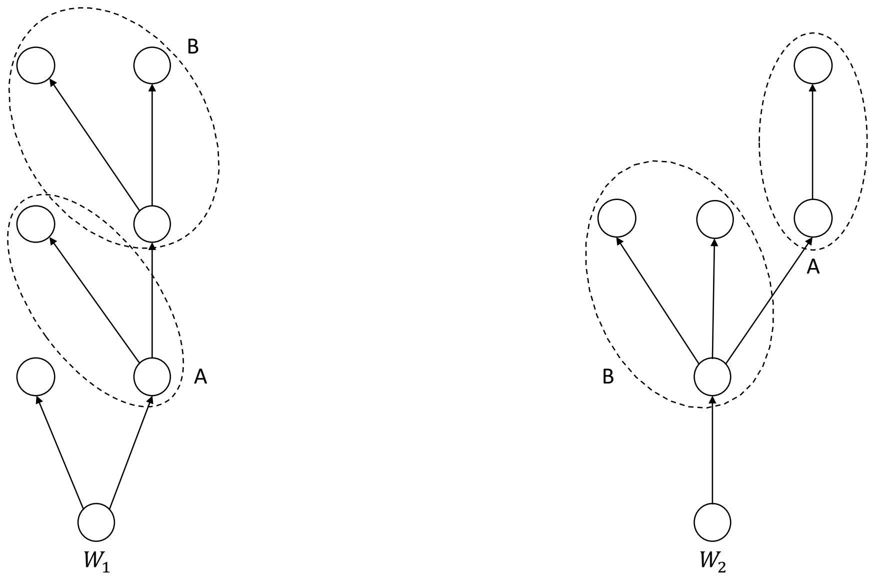


Figure 5.3: Cross-dependency of common-parts A and B in workflows W_1 and W_2

Chapter 6

Related Work

Most Workflow Management Systems (WMS) are described in the report D5.1 [19]. In this chapter we discuss only works that engage in optimization.

Workflow management systems have emerged in order to provide easy-to-use specification of tasks that have to be performed during data analysis. An essential problem that these systems need to solve is the combination of various tools for workflow execution and optimization over multiple engines into a single research analysis / system. The field of workflow management is a relatively new field of research, but there are already some promising results.

The HFMS system [15] builds on top of previous work on multi-engine execution optimization [16]. This research focuses on optimization and execution across multiple engines. The design of flows in HFMS is agnostic to a physical implementation. Data sets need not be bound to a data store, and operators need not be bound to an execution engine. HFMS handles flows as DAGs encoded in xLM, a proprietary language for expressing data flows.

Work related to HFMS [14] and [16] focuses on optimizing flows for several objectives: performance, fault-tolerance and freshness over multiple execution engines. Optimization is defined as a search space problem. These states are obtained by a large number of possible transitions: swap, factorize, distribute, compose, decompose, partition, add recovery point, replicate, parallelization, function shipping, data shipping, etc. Due the vastness of the state space, some techniques / heuristics are proposed to prune it. To assess the cost of states functions are defined both for operations (each vertex in a flow graph) and transitions. The work in [17] describes the process of construction of cost functions in more detail, and shows how they can be defined by engineer or during test runs or using micro-benchmarks. The experiments demonstrate the feasibility of their technique by comparing the heuristic solutions to those found by exhaustive search.

Thus, a significant difference of the HFMS system and related works from our research is that we are tackling and aim to propose solutions for a wider statement of the problem: the development of WMT in ASAP aims to manage workflows that have a variety of heterogeneous formats and can be processed on a variety of heterogeneous engines that may be, furthermore, distributed. Beyond this, the main difference of our research with HFMS and related works is that the latter focuses only on single-workflow optimization, whereas the first will focus also on multi-workflow optimization.

Chapter 7

Summary

This document describes the first version of WMT. This includes the declaration semantics of the workflow, design, analysis and optimization modules. Furthermore, we depict the work of WMT on specific use cases from the telecommunication and web analytics domains.Finally, we make an initial discussion on multi-workflow optimization.

Bibliography

[1] R. Bertoldi. Wp 9 - applications: Telecommunication data analytics. D9.2 Use Case Requirements, February 2015.

[2] Coffeescript. `http://coffeescript.org/`.

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

[4] Hypertext markup language. `http://www.w3.org/TR/html/`.

[5] Guy Jacobson and Kiem-Phong Vo. Heaviest increasing/common subsequence problems. In *Proceedings of the Third Annual Symposium on Combinatorial Pattern Matching*, CPM '92, pages 52–66, London, UK, UK, 1992. Springer-Verlag.

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

[7] Javascript. `https://www.javascript.com/`.

[8] D. Tsoumakos K. Doka, N. Papailiou et al. Wp 3 - intelligent, multi-engine resource scheduling platform. D3.2 IReS Platform v.1, August 2015.

[9] Verena Kantere and Maxim Filatov. A framework for big data analytics. In *Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering, Yokohama, Japan, July 13-15, 2015*, pages 125–132, 2015.

[10] Nginx. `http://nginx.org/`.

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

[12] Python. `https://www.python.org/`.

[13] Philippe Rigaux. Wp 8 - applications: Web content analytics. D8.2 Use Case Requirements, February 2015.

[14] A. Simitsis, K. Wilkinson, U. Dayal, and M. Castellanos. Optimizing etl workflows for fault-tolerance. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 385–396, March 2010.

[15] 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.

[16] 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.

[17] Alkis Simitsis, Kevin Wilkinson, and Petar Jovanovic. xpad: A platform for analytic data flows. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 1109–1112, New York, NY, USA, 2013. ACM.

[18] Robert Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1972.

[19] M. Filatov V. Kantere. Wp 5 - adaptive data analytics. D5.1 Workflow Management Model, February 2015.

[20] I-Hsuan Yang, Chien-Pin Huang, and Kun-Mao Chao. A fast algorithm for computing a longest common increasing subsequence. *Inf. Process. Lett.*, 93(5):249–253, March 2005.