Adaptive Scalable Analytics in Multi-Engine Environments

Verena Kantere
Associate Professor
School of Electrical Engineering and and Computer Science
University of Ottawa

Scalable and adaptive analytics

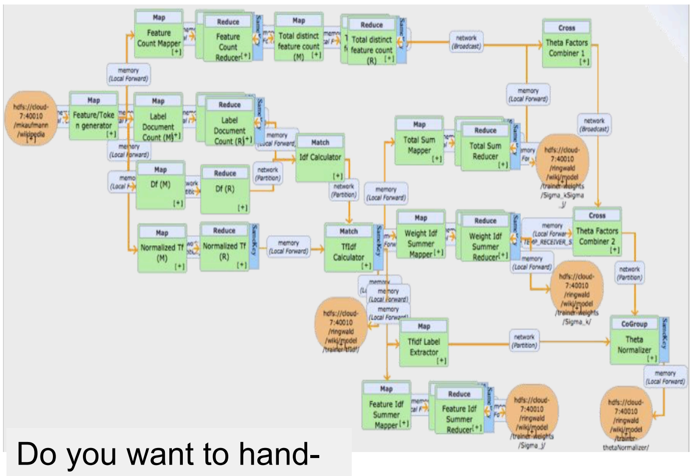
Motivation:

- Big Data: Exabytes... and growing!
- Analytics: Create knowledge wealth from existing data
- Big impact: Technology, Science, Economics, Medicine, Society etc

Challenges:

- Multiple engines, multiple data stores, many different people
- Applications connect multiple components, complex workflows
- Applications are difficult to construct, maintain, manage, optimize, execute, understand, schedule etc.

Why is automatization needed?



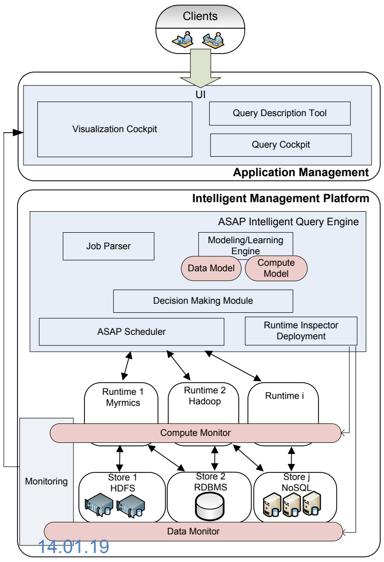
Do you want to hand-tune that?

Optimization of Workflows

- "At high-level" - performance depends on the experience of the designer
- "At low level" - execute workflow as it is; hopefully, the optimizer of the DBMS would improve the performance
- But what can be done "in the middle"?:
  - optimization of specific workflow parts
  - optimization of the whole workflow

The ASAP system

Adaptive Scalable Analytics Platform



FP7-ICT-2013-11, `Scalable data analytics' call, started March 2014, UniGe budget 535'600 €
Finished with evaluation "EXCELLENT"!

- Fully automated, highly customizable system
- Development and execution of arbitrary data analytics queries
- Large heterogeneous data store

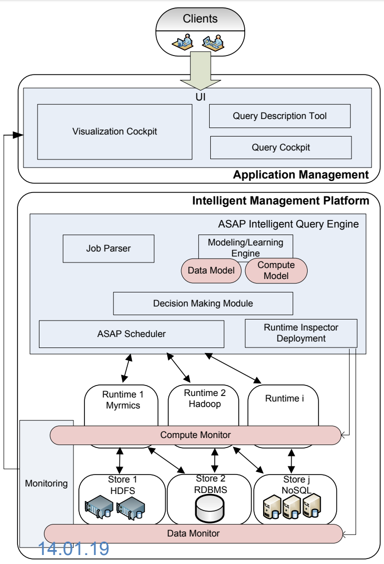
It offers:

- **A general-purpose task-parallel programming model**
  - Easy development of complex, irregular datacenter queries and applications
- **A modeling framework**
  - Consider type, location and size of data, type of computation, and resources
  - Decide on store, execution pattern and runtime machine
- **A unique adaptation methodology**
  - Calibrate queries and workflows
  - See intermediate results

14.01.19 Verena Kantere 5

The ASAP system

Adaptive Scalable Analytics Platform



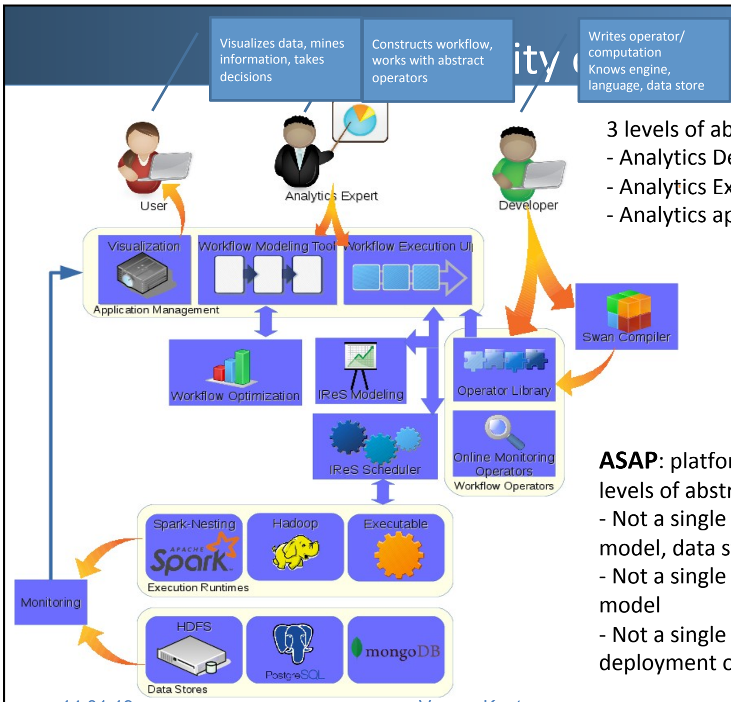
FP7-ICT-2013-11, `Scalable data analytics' call, started March 2014, UniGe budget 535'600 €
Finished last week with evaluation "EXCELLENT"!

- Fully automated, highly customizable system
- Development and execution of arbitrary data analytics queries
- Large heterogeneous data store

It offers:

- **A general-purpose task-parallel programming model**
  - Easy development of complex, irregular datacenter queries and applications
- **A modeling framework**
  - Consider type, location and size of data, type of computation, and resources
  - Decide on store, execution pattern and runtime machine
- **A unique adaptation methodology**
  - Calibrate queries and workflows
  - See intermediate results

14.01.19 Verena Kantere 6



3 levels of abstraction
- Analytics Developer
- Analytics Expert
- Analytics application user

ASAP: platform for managing all levels of abstraction
- Not a single optimal data model, data store
- Not a single optimal computing model
- Not a single optimal deployment on resources

Platform for Analytics Workflows (PAW)

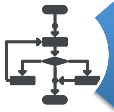
Workflow Model
How to accommodate users with different expertise?

Single Workflow Optimization
How to change the workflow to accelerate execution?

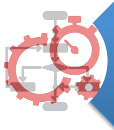
Multiple Workflow Optimization
How to execute workflows in a joint manner?

Workflow Recalibration
How to change workflows while they are executing?

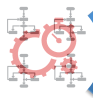
Platform for Analytics Workflows (PAW)



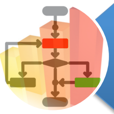
Workflow Model
How to accommodate users with different expertise?



Single Workflow Optimization
How to change the workflow to accelerate execution?



Multiple Workflow Optimization
How to execute workflows in a joint manner?



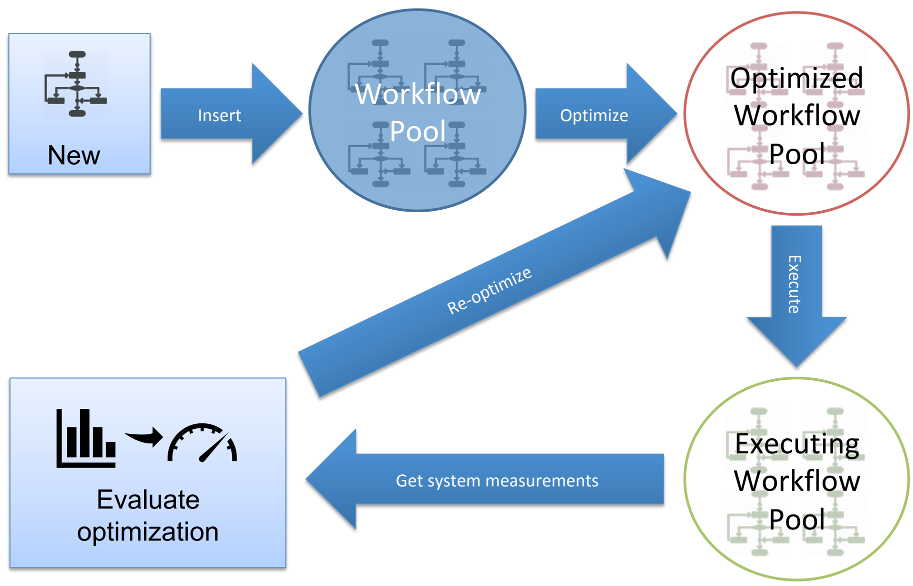
Workflow Recalibration
How to change workflows while they are executing?

Workflow management

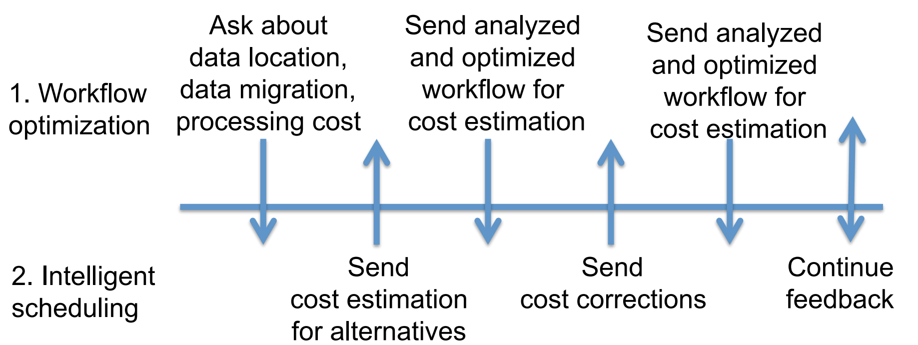
Creation → Analysis → Optimization

- A workflow is created by a user
- A workflow is analyzed
  - execution semantics are specified and
  - augmentation with associative tasks and task dependencies
- A workflow is optimized

Workflow management cycle



Two-stage optimization



Both levels need to do scheduling and optimization at different granularities
Feedback when:

- Schedule deviates from goal (2 ➔ 1)
- Changes in running workflows – new workflows or calibration (1 ➔ 2)

Workflow model

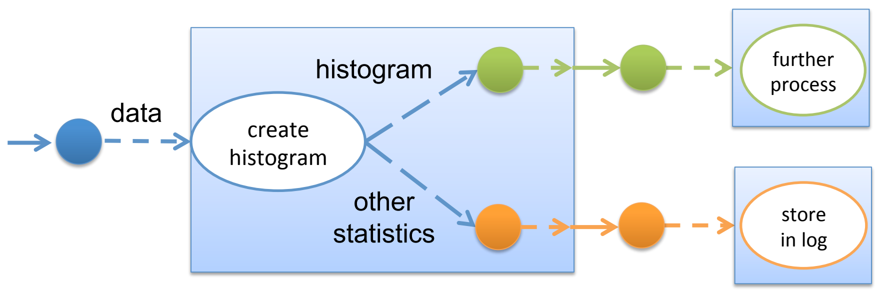
A workflow is a graph with vertices and edges

The workflow model:

- Enables the expression of application logic by users with various roles and expertise
- By separating task functionalities and task dependencies
- Allowing the specification or the abstraction of execution semantics

Vertices

- A vertex corresponds to a set of tasks
- A task corresponds to an Input, an Output and an Operator



Edges

- An edge corresponds to a pair of an input and an output.
- The input and the output are pairs of data and some metadata.
- The input and output of tasks are defined independently of the inputs and outputs of edges

Operators

- Operators are the core part of tasks
- They are user-defined or instantiated on templates
- Basic operators are formally defined and complex ones are stored procedures
- Metadata of operators are expressed in JSON
- The operators can be written with the programming language developed in ASAP

Examples of operators

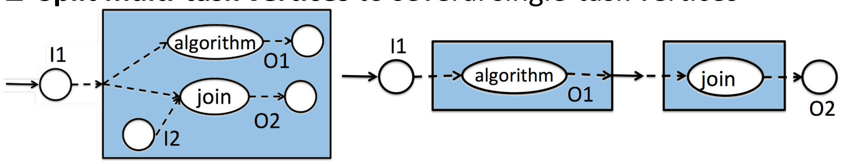
- $O(select; I) = \{r \mid r \in I \wedge SelectPredicate(r)\}$
- $O(calc; I) = \{r \cup \{attr : value\} \mid r \in I \wedge value := CalcExpression(r)\}$
- $O(join; I_1; I_2) = \{ t \cup s \mid t \in I_1 \wedge s \in I_2 \wedge JoinPredicate(t \cup s) \}$

Workflow analysis

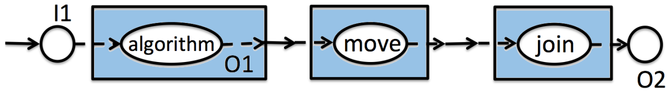
- **Validate consistency:**

A workflow is checked for cycles and correspondence of metadata of adjacent vertices

- **Split multi-task vertices** to several single-task vertices

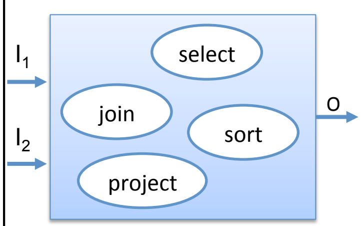


- **Augment the workflow with associative tasks** that convert data flow, convert the data format, move data etc

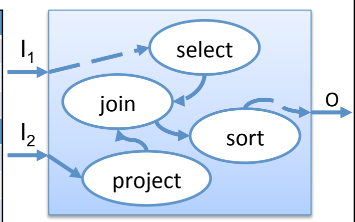


Splitting multi-task vertices

- Look for such case of multi-task vertex in the history
  - If exist then split that vertex, if not:
- Compare metadata of input and output for all pairs of tasks
- Find possible links between tasks
- Propose variants of tasks linkage to user
- Save chosen linkage into the history



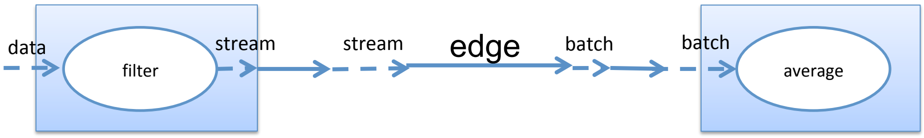
	schemas	
I_1	a_1, a_2	
I_2	b_1, b_2, b_3	
O	a_1, a_2, b_2	
tasks:	input	output
select	a_1, a_2	a_1, a_2
join	$a_1, a_2; b_1, b_2$	a_1, a_2, b_2
project	b_1, b_2, b_3	b_1, b_2
sort	a_1, a_2, b_2	a_1, a_2, b_2



Execution semantics of edges

- Edges with incompatible input/output metadata are substituted by associative triples:
  - An associative triple is a new vertex with an incoming and an outgoing edge. It holds a new task that changes the metadata of an edge
- Associative tasks may perform: scheduling, change of availability, or cleaning

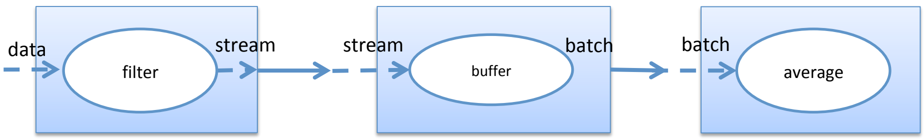
Scheduling example:



Execution semantics of edges

- Edges with incompatible input/output metadata are substituted by associative triples:
  - An associative triple is a new vertex with an incoming and an outgoing edge. It holds a new task that changes the metadata of an edge
- Associative tasks may perform: scheduling, change of availability, or cleaning

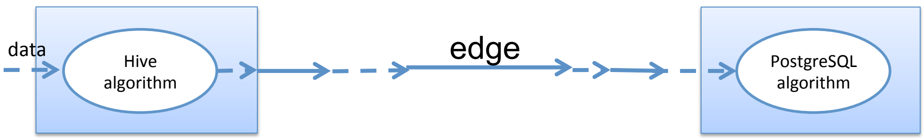
Scheduling example:



Execution semantics of edges

- Edges with incompatible input/output metadata are substituted by associative triples:
  - An associative triple is a new vertex with an incoming and an outgoing edge. It holds a new task that changes the metadata of an edge
- Associative tasks may perform: scheduling, change of availability, or cleaning

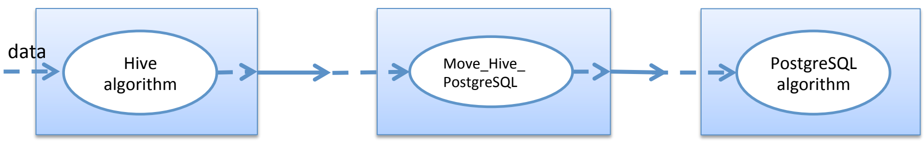
Availability example:



Execution semantics of edges

- Edges with incompatible input/output metadata are substituted by associative triples:
  - An associative triple is a new vertex with an incoming and an outgoing edge. It holds a new task that changes the metadata of an edge
- Associative tasks may perform: scheduling, change of availability, or cleaning

Availability example:



Towards workflow optimization

- A workflow is optimized so that it can be executed more efficiently that originally designed
- The final outputs should remain the same after optimization
- Optimization is performed employing transitions

Operator characteristics

Workflow optimization can be performed selectively depending on characteristics of operators:

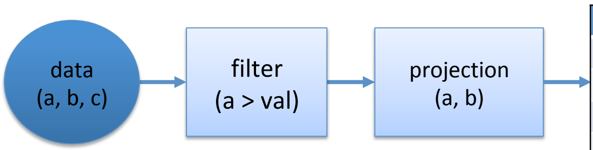
- **Blocking** operators require knowledge of the whole dataset
- **Non-blocking** operators that process each tuple separately
- **Restrictive** operators output smaller than incoming data volume

Operator	Blocking	Non-blocking	Restrictive
Filter		x	x
Calc		x	
groupBy Sort	x		
Wind_DataFilter	x		
Wind_PeakDet	x		
Wind_KMeans	x		
Wind_Stereotype_ Classification	x		
Wind_Distribution_ Computation	x		
Wind_User_Profiling	x		
Filter_Join	x		x
Filter_Calc		x	x
TF-IDF	x		
lr_train	x		
lr_classify	x		
Move_Hive_Postgres	x		
Move_Postgres_Hive	x		
w2v_train	x		
w2v_vectorize	x		
grep	x		
Join	x		
Join4	x		
Left_Outer_Join	x		
Projection		x	x

Operator characteristics cont'd

In order to apply transitions, apart from the input and output schema, each task is characterized by the following schemas:

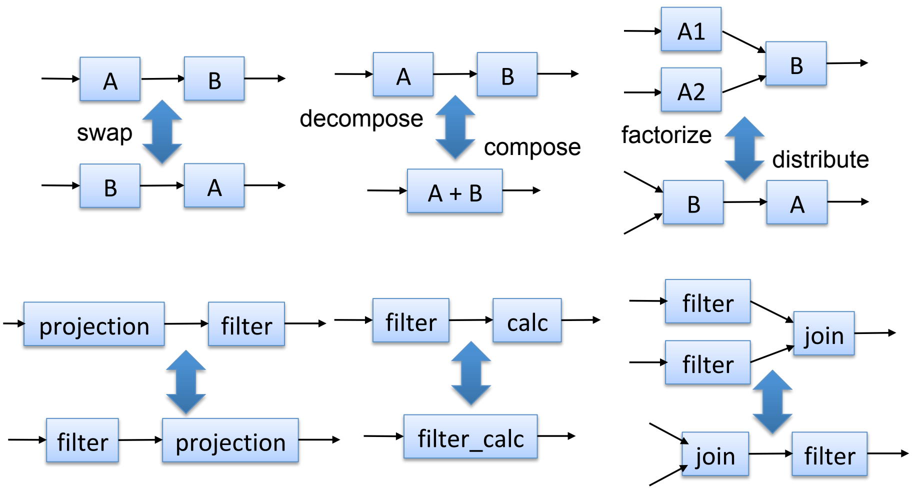
- **Functionality schema** (fs): is a list of attributes that are processed by the task. They are a subset of (the union of) the input schemas
- **Generated schema** (gs): is a list of all the output attributes that are generated by the task
- **Projected-out schema** (pos): is a list of attributes that belong to the input schema, but are not output by the task



schemas	filter	projection
functionality	a	∅
generated	∅	∅
projected-out	∅	c
input	a,b,c	a,b,c
output	a,b,c	a,b

Optimization via graph reconfiguration

Transitions generating equivalent workflow versions:



Functionality of transitions

Swap

- Allows for pushing highly selective operators towards the root of the workflow
- Swapping is not relational algebra pushing down because of the presence of functions

Composition and decomposition

- Allow for the replacement of complex operators with simpler ones and vice versa
- Create optimization opportunities adaptive to the environment: available machines, engines, current workload, size of data etc

Factorization and distribution

- Factorization allows for the replacement of multiple identical operators with one performed on the sum of the datasets: operation is performed only once on an aggregated dataset
- Distribution allows for the opposite: it parallelizes execution and/or reduces the input data size

14.01.19 Verena Kantere 27

Applicability of transitions

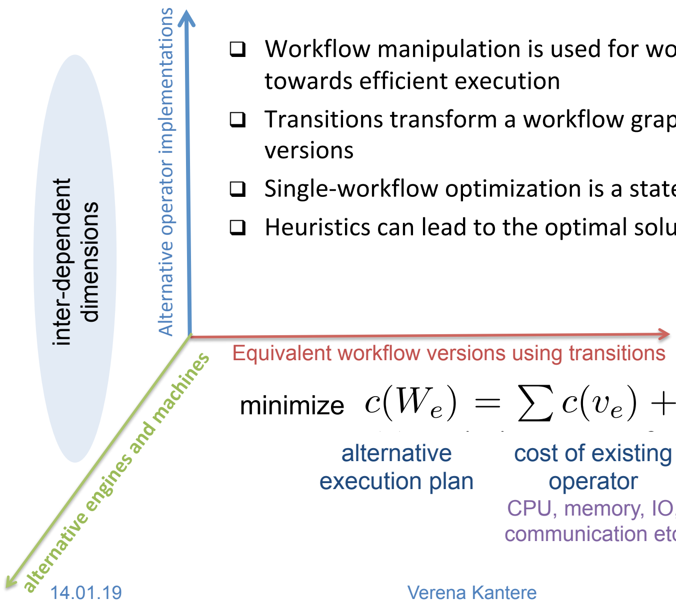
Applicability of transitions in based on the schemas

swap	filter	calc	join	filter_calc	filter_join	projection
filter	✓	✓	✓	✓	✓	If $filter.fs \cap projection.pos = \emptyset$
calc	If $calc.gs \cap filter.fs = \emptyset$	If $calc1.gs \cap calc2.fs = \emptyset$	If $calc.gs \cap join.fs = \emptyset$	If $calc.gs \cap filter\_calc.fs = \emptyset$	If $calc.gs \cap filter\_join.fs = \emptyset$	If $calc.fs \cap projection.pos = \emptyset$
join	If $filter.fs \subset join.i1s$ or $filter.fs \subset join.i2s$	If $calc.fs \subset join.i1s$ or $calc.fs \subset join.i2s$	If $join1.fs \subset join2.i1s$ or $join1.fs \subset join2.i2s$	If $filter\_calc.fs \subset join.i1s$ or $filter\_calc.fs \subset join.i2s$	If $filter\_join.fs \subset join.i1s$ or $filter\_join.fs \subset join.i2s$	If $join.fs \cap projection.pos = \emptyset$ and $projection.pos \subset join.i1s$ or $projection.pos \subset join.i2s$
filter_calc	If $filter\_calc.gs \cap filter.fs = \emptyset$	If $filter\_calc.gs \cap calc.fs = \emptyset$	If $filter\_calc.gs \cap join.fs = \emptyset$	If $filter\_calc1.gs \cap filter\_calc2.fs = \emptyset$	If $filter\_calc.gs \cap filter\_join.fs = \emptyset$	If $filter\_calc.fs \cap projection.pos = \emptyset$
filter_join	If $filter.fs \subset filter\_join.i1s$ or $filter.fs \subset filter\_join.i2s$	If $calc.fs \subset filter\_join.i1s$ or $calc.fs \subset filter\_join.i2s$	If $join.fs \subset filter\_join.i1s$ or $join.fs \subset filter\_join.i2s$	If $filter\_calc.fs \subset filter\_join.i1s$ or $filter\_calc.fs \subset filter\_join.i2s$	If $filter\_join1.fs \subset filter\_join2.i1s$ or $filter\_join1.fs \subset filter\_join2.i2s$	If $filter\_join.fs \cap projection.pos = \emptyset$ and $projection.pos \subset filter\_join.i1s$ or $projection.pos \subset filter\_join.i2s$
projection	✓	✓	✓	✓	✓	✓

Applicability table for swap and other operators

14.01.19 Verena Kantere 28

Workflow optimization



- Workflow manipulation is used for workflow optimization towards efficient execution
- Transitions transform a workflow graph into equivalent versions
- Single-workflow optimization is a state search problem
- Heuristics can lead to the optimal solution quickly

minimize $c(W_e) = \sum c(v_e) + \sum c(a_e)$

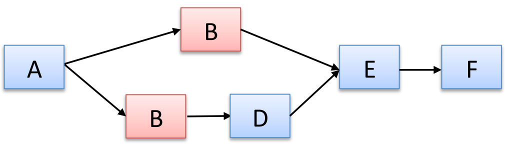
- alternative execution plan
- cost of existing operator: CPU, memory, IO, communication etc
- cost of associative operator: data shipping, initializing engine, bandwidth etc

Improving search performance

Using heuristics:

- Composing is used where it is applicable, it provides more opportunities for micro-optimization on engines

- Finding of homologous tasks accelerates the generation of a search space, because it eliminates unnecessary attempts of factorizing

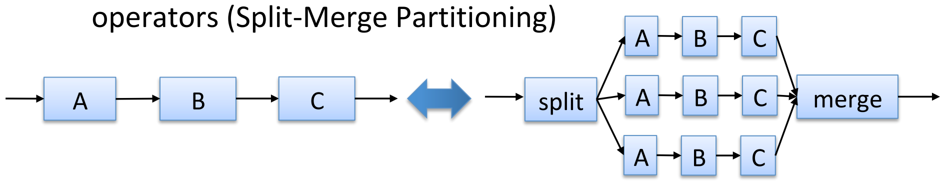


Pruning the search space

Using heuristics:

- Restrictive operators are moved to the root of the workflow to reduce the data volume

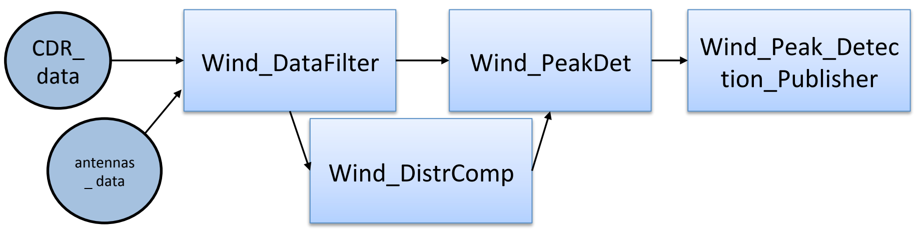
- Non-blocking operators are placed together and separately from blocking operators in order to parallelize non-blocking operators (Split-Merge Partitioning)



Heuristics may lead to near-optimal version in absence of some cost metrics!

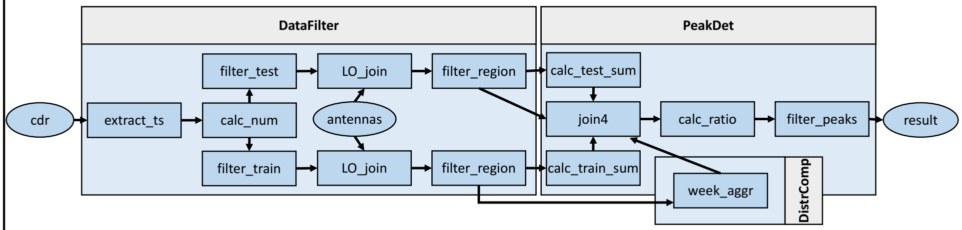
Telecommunication analytics application

- Analysis of telecommunication data:
  - detection peaks in mobile calls

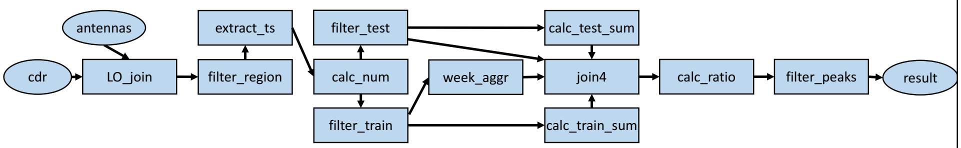


- It involves processing anonymised Call Detail Records (CDR) data for Rome, from 01/01/2015 until 31/12/2015
  - *CDR_data(call_id, timestamp, user_id, antenna_id)*
  - *antennas(antenna_id, region_id)*

Alternative and optimized workflow



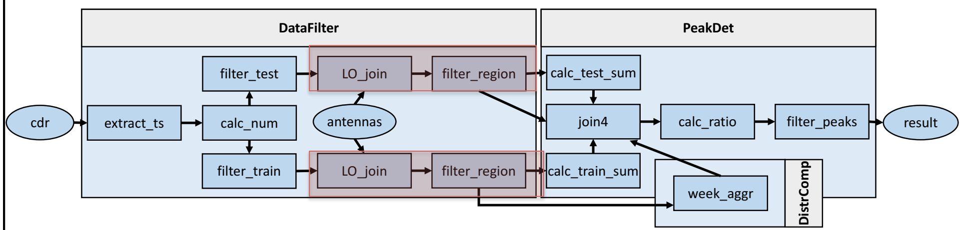
Workflow on PostgreSQL



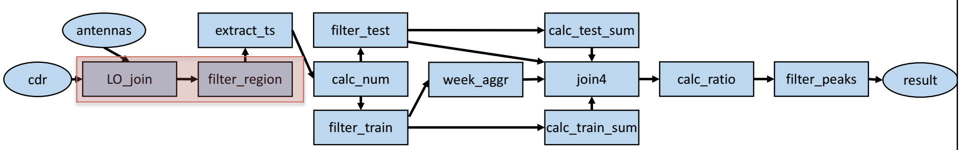
Optimized workflow on PostgreSQL

In the optimized workflow:
- **LO_join and filter_region** are swapped with **filter_test** and **filter_train**
- **LO_join and filter_region** are factorized over **calc_num**
- **HS** and **ES** produce the same version: HS composes **LO_join** and **filter_region**; the composed is a restrictive operator pushed towards the root and then decomposed

Alternative and optimized workflow



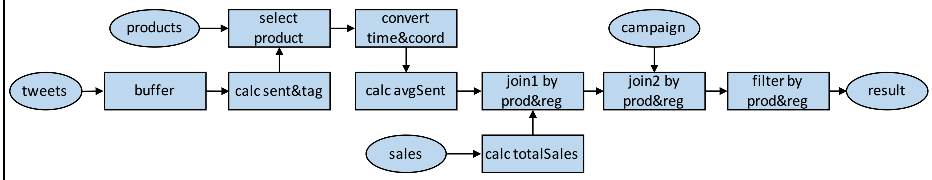
Workflow on PostgreSQL



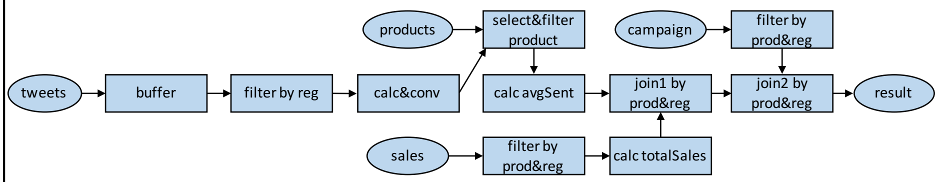
Optimized workflow on PostgreSQL

In the optimized workflow:
- **LO_join and filter_region** are swapped with **filter_test** and **filter_train**
- **LO_join and filter_region** are factorized over **calc_num** and **extract_ts**
- **HS** and **ES** produce the same version: HS composes **LO_join** and **filter_region**; the composed is a restrictive operator pushed towards the root and then decomposed

Example use case from marketing



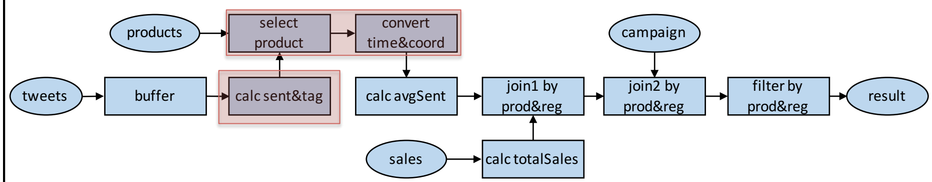
Analyzed version of the original workflow



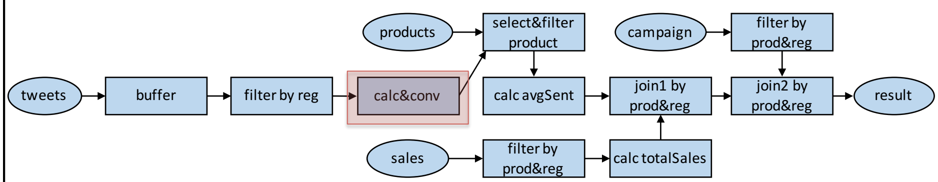
Optimized version of the analyzed workflow

In the optimized workflow:
- **select_product** and **convert_time&coord** are swapped
- **convert_time&coord** and **calc_sent&tag** are composed
- **filter_by_prod&reg** is broken down to **filter_by_prod&reg** and **filter_by_reg**
- **filter_by_reg** is pushed towards **tweets reviews**
- **filter_by_reg** is composed with **select_product**

Example use case from marketing



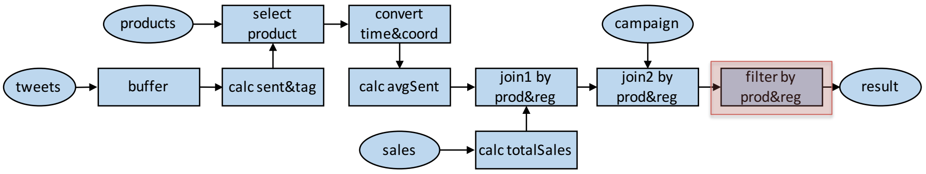
Analyzed version of the original workflow



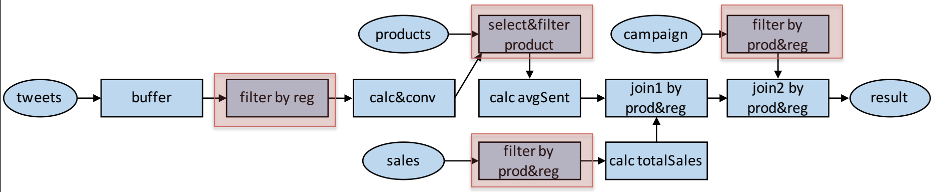
Optimized version of the analyzed workflow

In the optimized workflow:
- **select_product** and **convert_time&coord** are swapped
- **convert_time&coord** and **calc_sent&tag** are composed
- **filter_by_prod&reg** is broken down to **filter_by_prod&reg** and **filter_by_reg**
- **filter_by_reg** is pushed towards **tweets reviews**
- **filter_by_reg** is composed with **select_product**

Example use case from marketing



Analyzed version of the original workflow

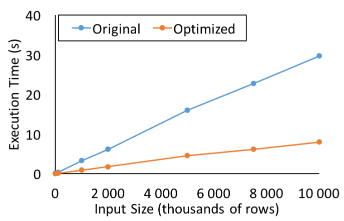


Optimized version of the analyzed workflow

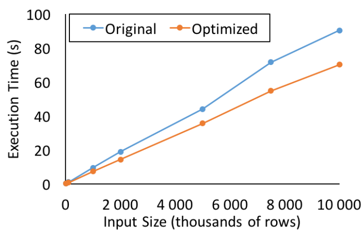
In the optimized workflow:

- **select_product** and **convert_time&coord** are swapped
- **convert_time&coord** and **calc_sent&tag** are composed
- **filter_by_prod&reg** is broken down to **filter_by_prod&reg** and **filter_by_reg**
- **filter_by_reg** is pushed towards **tweets reviews**
- **filter_by_reg** is composed with **select_product**

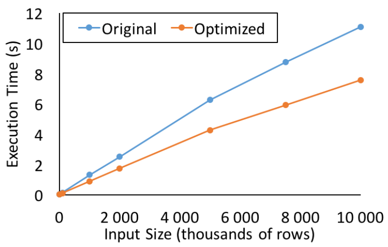
Experimental results for real workflows



Original and optimized marketing campaign workflow

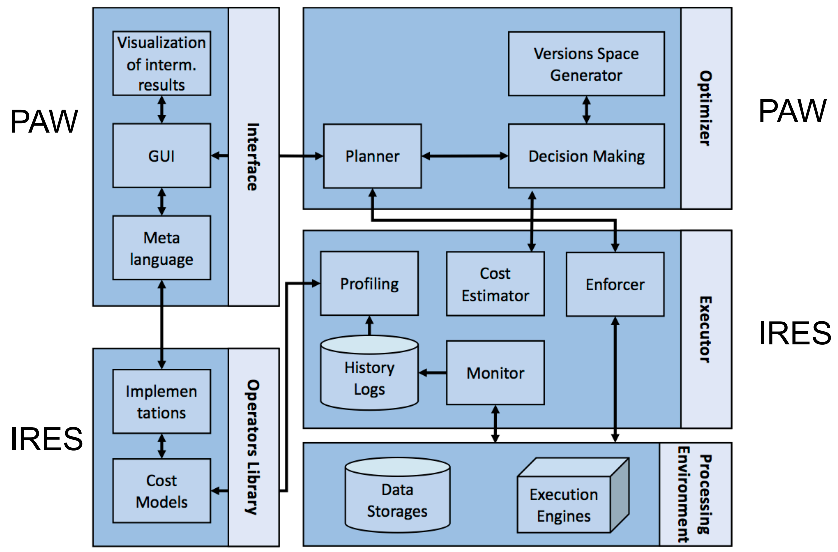


Original and optimized Peak Detection workflow



Original and optimized User Profiling workflow

The architecture of PAW and its interaction

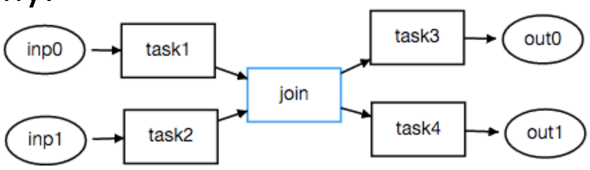


Benchmarking

- Benchmark produces synthetic workflows
- Synthetic workflows are based on graph patterns and "filled" with queries generated using TPC-DS.
- Experimental parameters:
  - the size of the workflow
  - the structure of the workflow
  - the percentage of operators of specific type
  - size of common part (testing MWO)
  - number of common parts (testing MWO)

Workflow graph patterns

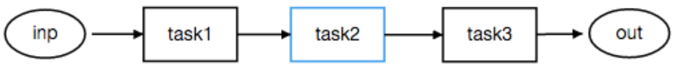
❑ Butterfly:



Butterflies are used to create ETL processes, typically:

- Left wing performs the extraction and transformation, and loads data to the body
- Body merges parallel data flows
- Right wing supports reporting and analysis – materializes views, creates reports

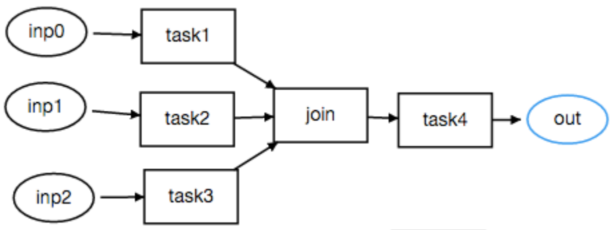
❑ Line:



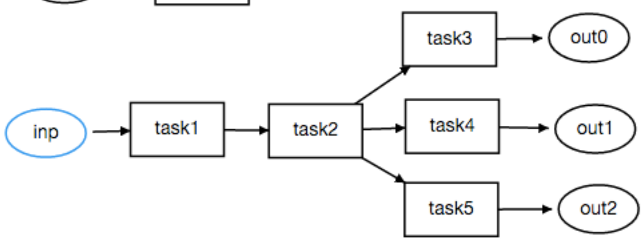
Lines are single data flows

Patterns cont'd

❑ Tree:



❑ Fork:



- Forks and trees are used to create memory-intensive workflows, by including sorting and aggregating operators
- Combined with lines they can be employed to study, also, pipelining

Benchmark details

- two tables: *web sales* and *customers* from TPC-DS
- 30+ query templates
- benchmark parameters:

Parameter	range	constant
Workflow size	10–200	20–50
Workflow structure		
butterfly	10–70%	25%
line	10–70%	25%
fork	10–70%	25%
tree	10–70%	25%
Operators		
blocking	0–100%	25–75%
non-blocking	0–100%	25–75%
restrictive	0–100%	25–75%

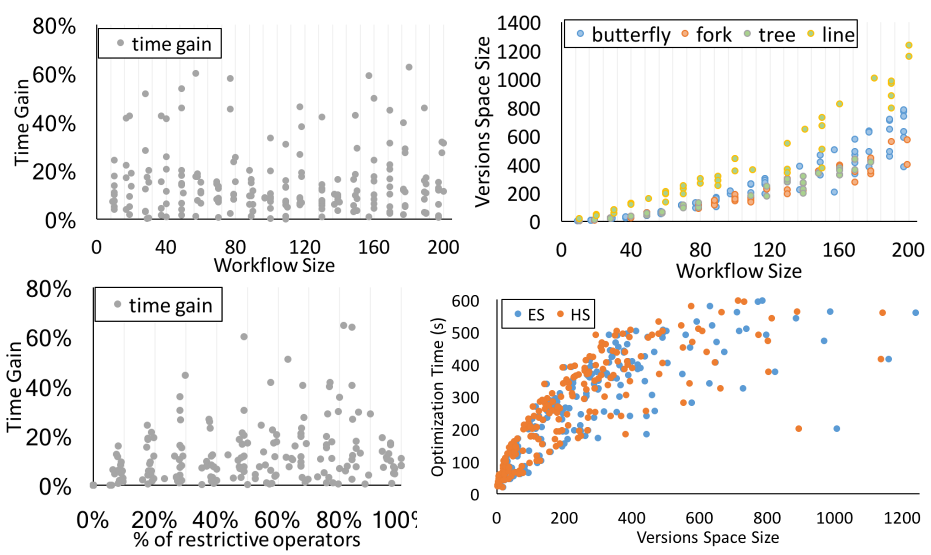
```
define YEAR=random(1996,2001,uniform);
define RANGE=random(1,4,uniform)
define LISTPRICE=ulist(random(0,190,uniform),2);
define _LIMIT=random(1,rowcount(web_sales),uniform);

[_LIMITA] select [_LIMITB] *
 from web_sales
where YEAR(ws_sold_date) between [YEAR]-[RANGE]
       and [YEAR]-[RANGE]
  and ws_list_price between [LISTPRICE.1]
       and [LISTPRICE.2]
order by ws_order_number
[_LIMITC];
```

- 300+ queries of four combinations of operator types: blocking and restrictive, non-blocking and restrictive, blocking and non-restrictive, non-blocking and non-restrictive

Questions answered in experiments

- How fast does the algorithm produce an optimized version of a workflow?
- What is the performance gain of the optimized version with respect to the performance of the original workflow?
- How large is the search space generated by the algorithms?
- What is the impact of workflow characteristics (workflow size, structure, percentage of blocking, non-blocking and restrictive operators, input data size)?
- Do the algorithms produce the same solutions?
- How does optimization cope with operators of agnostic cost?

Experimental results for synthetic workflows



Platform for Analytics Workflows (PAW)

Workflow Model

How to accommodate users with different expertise?

Single Workflow Optimization

How to change the workflow to accelerate execution?

Multiple Workflow Optimization

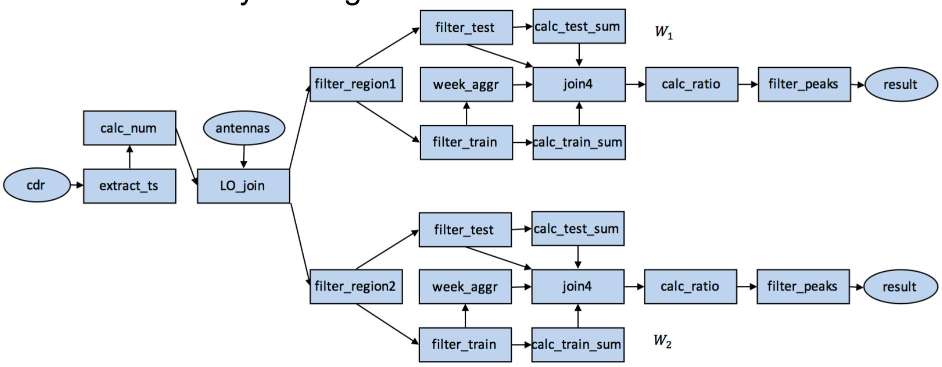
How to execute workflows in a joint manner?

Workflow Recalibration

How to change workflows while they are executing?

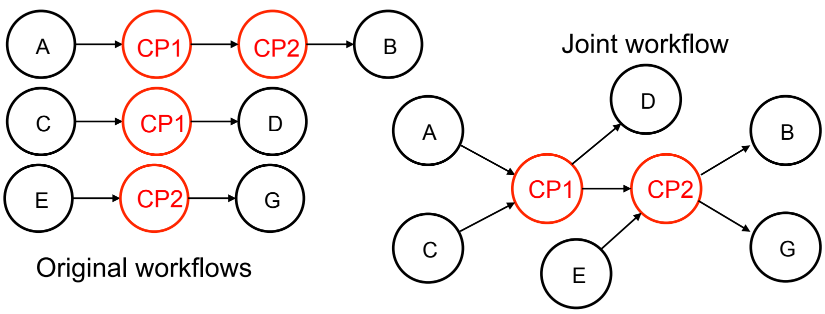
Motivating example

Joint workflow of two "Peak detection" workflows with different analysed regions

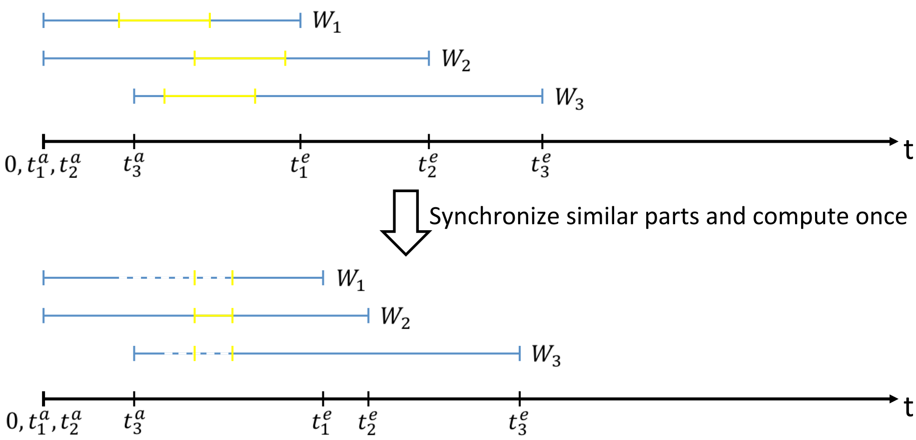


Multi-workflow optimization

- Our main approach is to find similar graph parts between workflows
  - Topological comparison: finding common sub-graphs
  - Tasks/metadata comparison: data scheme, operator details

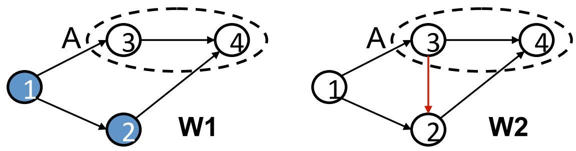


Multi-workflow optimization cont'd



Finding common parts

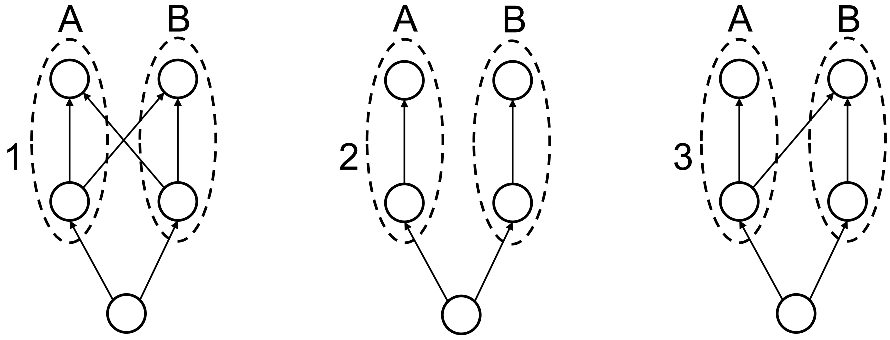
- Execution state ES(W):
  - some of the vertices are assumed to have been executed and no vertices are being executed
- Independently executable subgraph S w.r.t. ES(W):
  - S can be executed without executing any vertex in W \ {ES(W)}
- (Not) independently executable subgraph A:



The creation of a joint workflow W_o of a set *W* = {W_1, . . . , W_m} that have one common part CP, is possible if CP is independently executable for some execution state for every workflow in *W*

Combining by several common parts

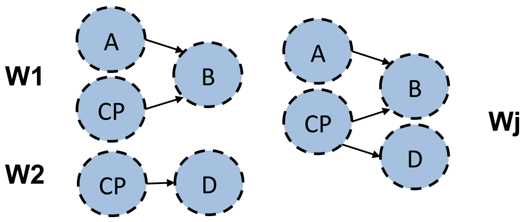
- Mutual arrangement of subgraphs A and B



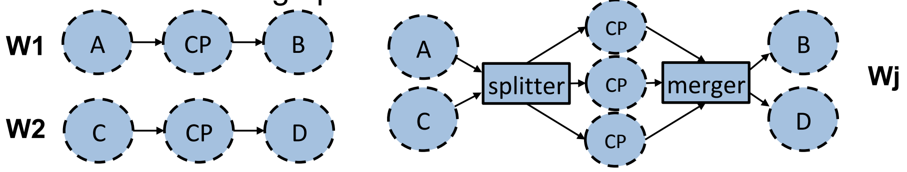
- Depending on their mutual arrangement in the set of workflows, a pair of common parts can be selected for the construction of the joint workflow or not.

Combining by a common part

- Common part at the beginning of workflows



- Common part in the middle of workflows consisting only of non-blocking operators



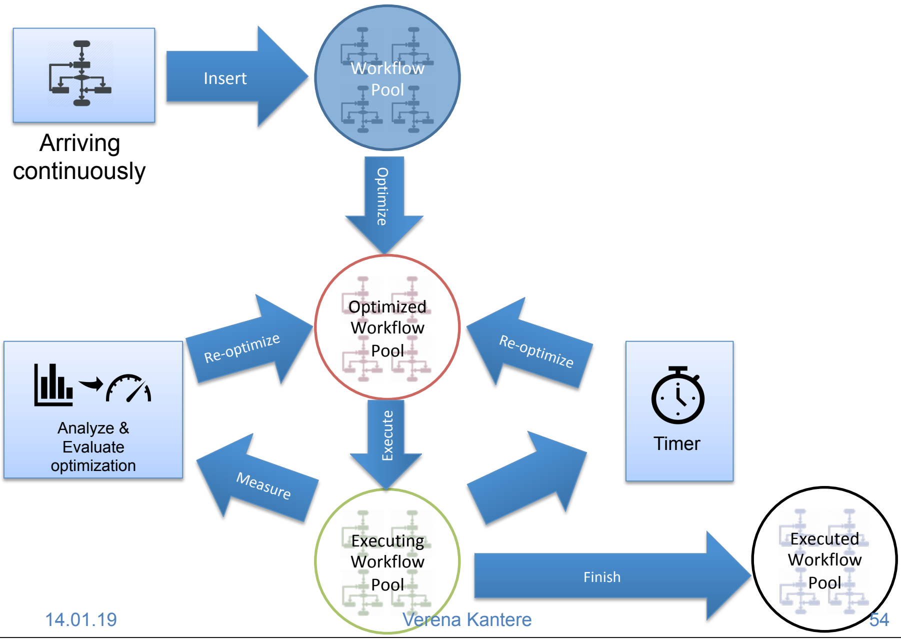
Execution cost for joint workflows

- The processing cost of of a joint workflow W_o of workflows $W = \{W_1, . . . , W_m\}$ with common parts $\{CP_1, . . . , CP_{nl}\}$ is:

$$C(W_1 o \ldots o\, W_m) = \sum_{i=1}^{m} C(W_i) - \sum_{i=1}^{n} ((l_i - 1)C(CP_i) - C(sync_i))$$

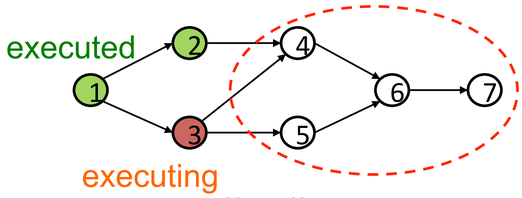
where l_i is the number of occurences of common part CP_i in *W* and $sync_i$ is the cost of syncronization of execution of common parts.

Workflow management cycle

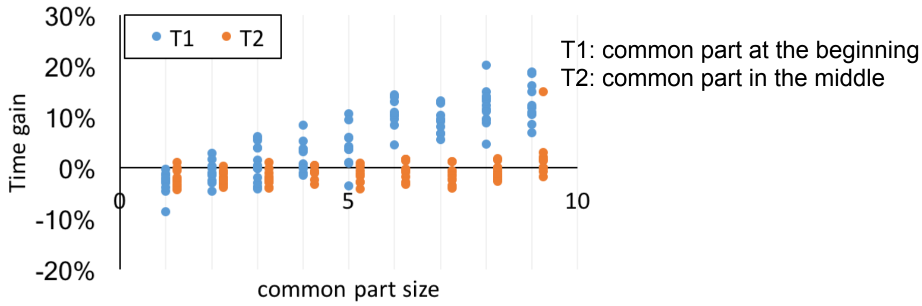


Online multi-workflow optimization

- Online multi-workflow optimization re-optimizes currently running workflows on each addition of a new workflow
- Current non-executed workflow parts are taken as an input
- Online multi-workflow optimization is done w/o aborting the execution of workflows
- If new optimized joint workflow is produced then PAW aborts current runs and executes re-optimized system of workflows
- As an improvement, we can estimate the remaining time of executing tasks. Then, based on this we decide to add a task to a partial workflow or not.



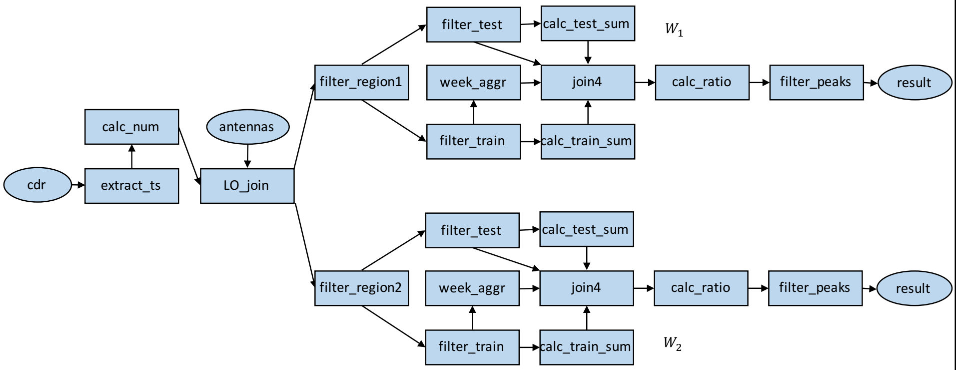
Results (multi-workflow optimization)



200 sets of workflows automatically generated of the following configuration: One common part of 1–10 nodes; Number of workflows in a set - 2–5; Workflow size - 20–50 vertices; Common part operators [blocking, non-blocking, restrictive] - [25–75%, 25–75%, 25–75%].

Results (multi-workflow optimization)

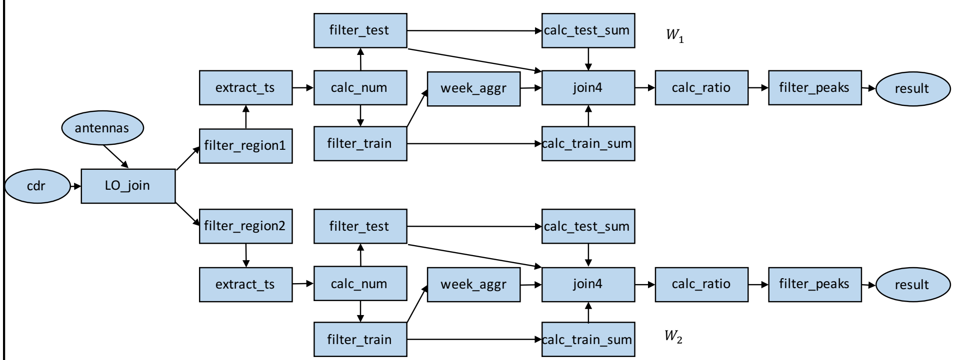
There is a total 12 regions in the input dataset CDR. In this run both workflows limit their analyzed area in **8** regions



Optimal joint workflow of two ‘Peak Detection’ workflows if total selectivity of filter_region1 and filter_region2 is low

Results (multi-workflow optimization)

There is a total 12 regions in the input dataset CDR. In this run both workflows limit their analyzed area in **4** regions



Optimal joint workflow of two ‘Peak Detection’ workflows if total selectivity of filter_region1 and filter_region2 is high

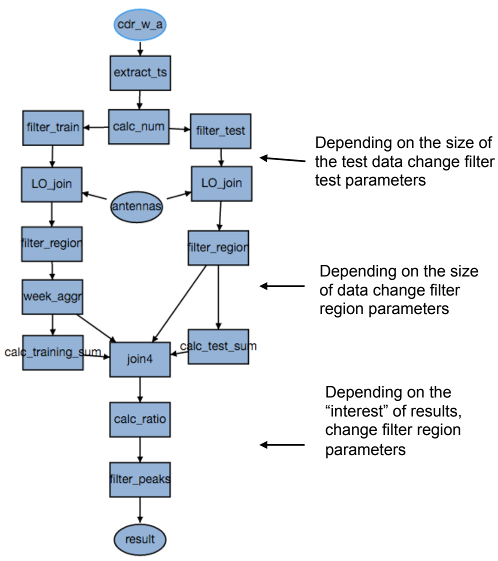
MWO also considers 3 single-vertex common parts: filter_test, filter_train and filter_peaks. But split-merge only increases the cost of processing.

Workflow recalibration

- It enables the analytics to change the workflow by altering the task parameters or infusing new tasks
- It entails the following requirements:
  - Enable access to intermediate results
  - Enable workflow changes at runtime
  - Avoid repeated computations

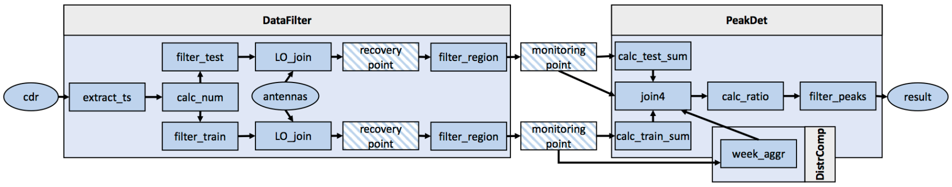
Workflow recalibration

- It enables the analytics to change the workflow by altering the task parameters or infusing new tasks
- It entails the following requirements:
  - Enable access to intermediate results
  - Enable workflow changes at runtime
  - Avoid repeated computations



Motivating example

Peak detection with recovery and monitoring points



Manual technique of recalibration

A technique based on recovery and monitoring points:

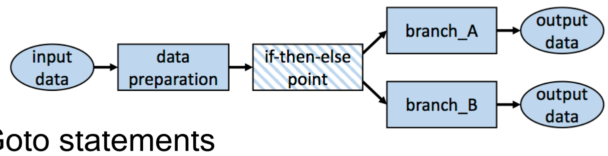
- observe intermediate results on monitoring points
- re-run changed workflow from recovery point

- ❑ Recalibration points are displayed only in PAW, and are not sent to IRES
- ❑ Using these points, PAW performs recalibration: decides which parts of the workflow and when to execute or re-execute
- ❑ Three basic monitoring operators, for the visualization of: numerical, categorical and geographical data

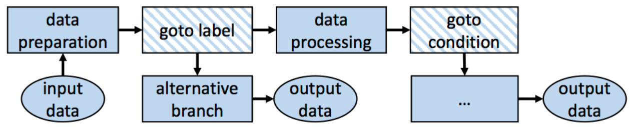
Automatic techniques of recalibration

A technique for automated re-calibration:

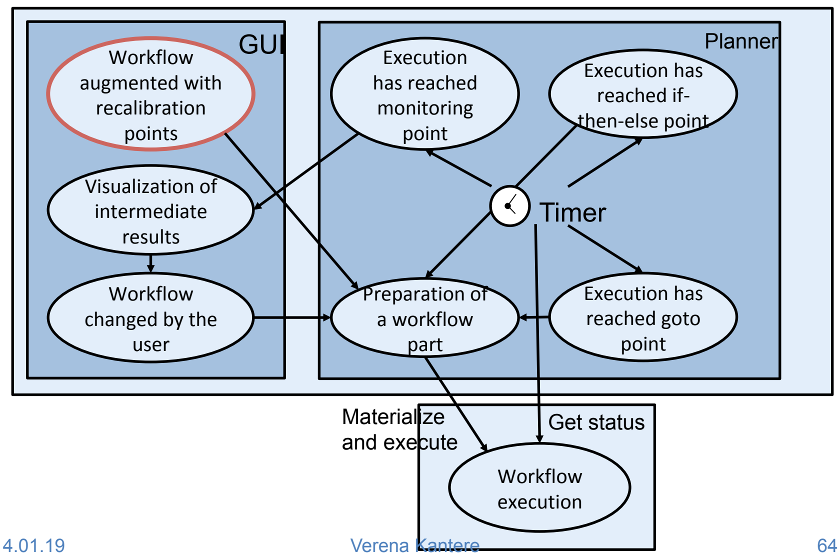
- Conditional statements - `if-then-else' constructions



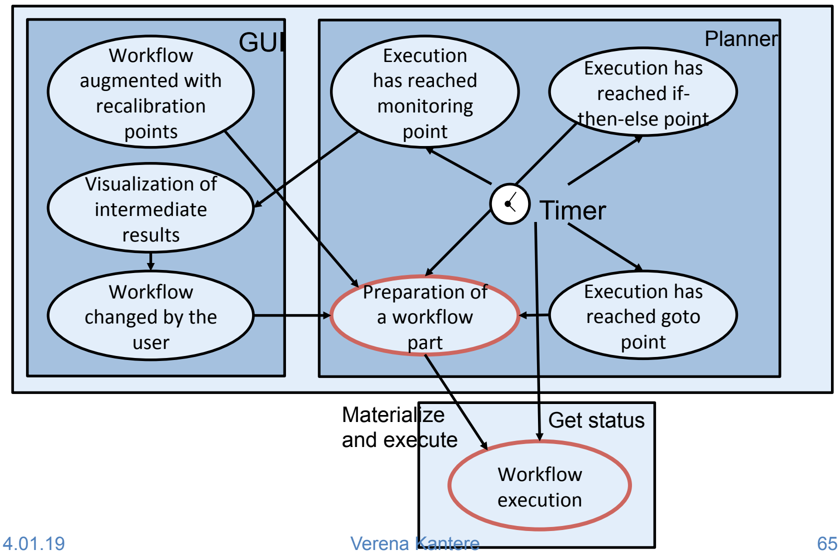
- Goto statements



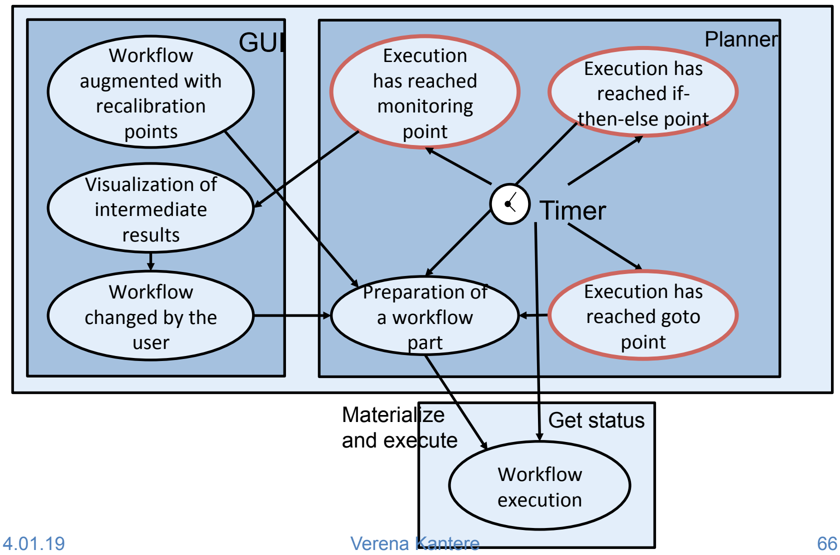
Recalibration process



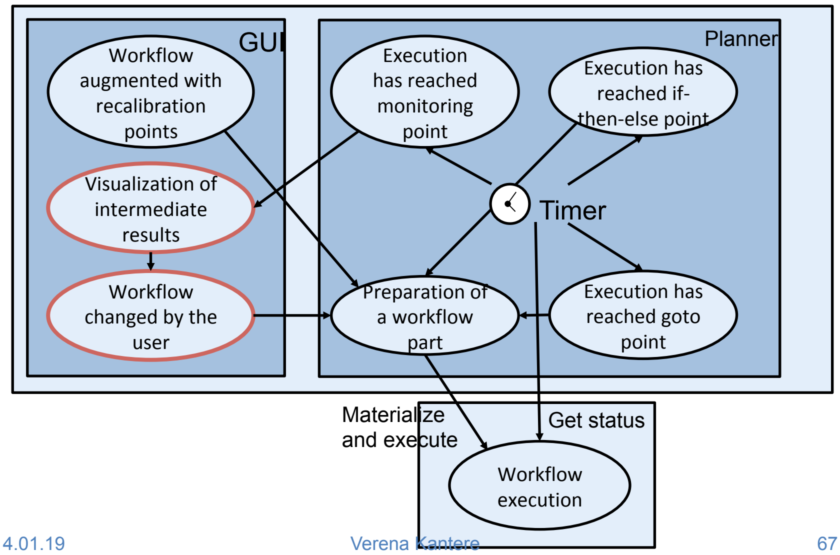
Recalibration process



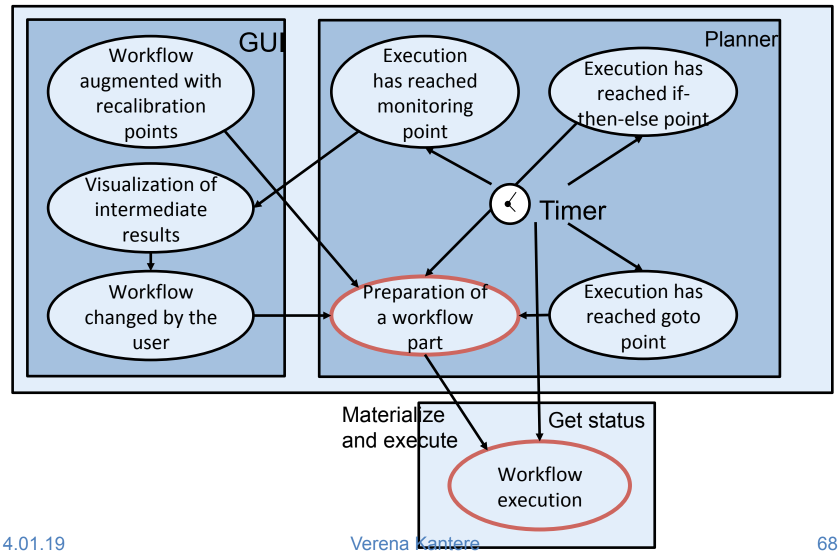
Recalibration process



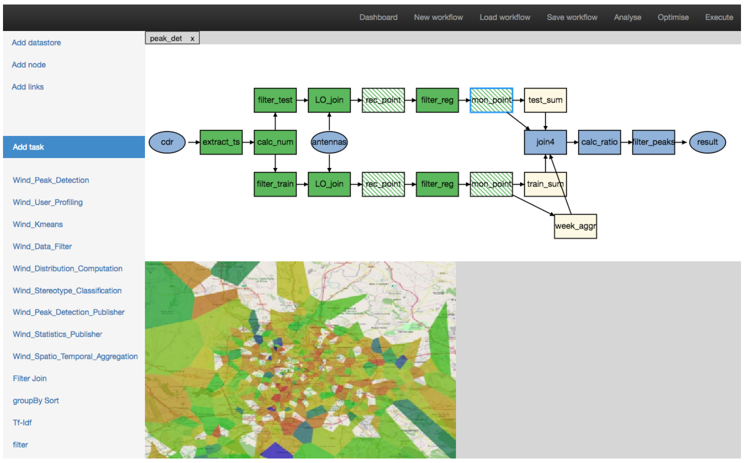
Recalibration process



Recalibration process



Screenshot of PAW



Monitoring intermediate results of 'Peak Detection' in PAW

Publications on PAW

1. V. Kantere and M. Filatov. Modelling processes of big data analytics. In WISE, 2015.
2. V. Kantere and M. Filatov. A framework for big data analytics. In C3S2E, 2015.
3. M. Filatov and V. Kantere. PAW: A Platform for Analytics Workflows. (Demo) in EDBT, 2016.
4. V. Kantere et al. Optimizing, Planning and Executing Analytics Workflows over Multiple Engines. In MEDAL, 2016.
5. M. Filatov and V. Kantere. Workflow Optimization in PAW. In ICDCS, 2017.
6. M. Filatov, V. Kantere. Multi-Workflow Optimization in PAW. (Demo) in EDBT, 2017.
7. M. Filatov, V. Kantere. (Tutorial on) Data Analytics in Multi-Engine Environments. In DASFAA, 2017.
8. M. Filatov, V. Kantere. (Tutorial on) Data Analytics in Multi-Engine Environments. In DAMDID, 2016.
9. M. Filatov, V. Kantere. Recalibration of Analytics Workflows. (Demo) in EDBT 2018.

Related work

- **Pegasus** (University of Southern California, ISI) (2001 – now)
- **HFMS**, **xPAD** (*HP Labs*) (2002 – ?)
- **Taverna** (University of Manchester, Cardiff University, University of Amsterdam) (2004 – now)
- **SQL++**, **FORWARD** (UCSD) (2010 – now)
- **Stratosphere** (TU Berlin) (2010 – 2015)
- **Apache Flink** (TU Berlin) (2014 – now)
- **Emma** (TU Berlin) (2015 – now)
- **BigDAWG** Polystore System (UofC, MIT, Intel) (2015 – now)
- **Rheem** (QCRI, HBKU) (2015 – now)
- **ASAP** (FORTH-ICS, UNIGE, ICCS, QUB, IMR, WIND, webLyzard) (2014– 2017)

Comments and questions?

vkantere@uottawa.ca