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
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

Finished last week with evaluation “EXCELLENT”!

Verena Kantere
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

- 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

1. Workflow optimization
   - Ask about data location, data migration, processing cost
   - Send analyzed and optimized workflow for cost estimation
   - Send analyzed and optimized workflow for cost estimation

2. Intelligent scheduling
   - Send cost estimation for alternatives
   - Send cost corrections
   - Continue feedback

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.

Diagram:
- Data is input.
- Create histogram.
- Other statistics.
- Further process.
- Store in log.
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

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

---

```
<table>
<thead>
<tr>
<th>task</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>select</td>
<td>$a_1$, $a_2$</td>
<td>$a_1$, $a_2$</td>
</tr>
<tr>
<td>$I_1$</td>
<td>$a_2$, $b_2$</td>
<td>$a_2$, $b_2$</td>
</tr>
<tr>
<td>$I_2$</td>
<td>$a_1$, $a_2$, $b_2$</td>
<td>$a_1$, $a_2$, $b_2$</td>
</tr>
<tr>
<td>join</td>
<td>$a_1$, $a_2$, $b_2$</td>
<td>$a_1$, $a_2$, $b_2$</td>
</tr>
<tr>
<td>project</td>
<td>$b_1$, $b_2$, $b_3$</td>
<td>$b_1$, $b_2$, $b_3$</td>
</tr>
<tr>
<td>sort</td>
<td>$a_1$, $a_2$, $b_2$</td>
<td>$a_1$, $a_2$, $b_2$</td>
</tr>
</tbody>
</table>
```

---

Verena Kantere
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:**

![Diagram of scheduling example](image)

14.01.19

Verena Kantere

19
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 than 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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Blocking</th>
<th>Non-blocking</th>
<th>Restrictive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Calc</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>groupby_sort</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wind_DataFilter</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wind_PeakDet</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wind_KMeans</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wind_Stereotype_Classification</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wind_Distribution_Computation</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wind_User_Profileing</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Filter_Calc</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>TF_IDF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lr_train</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lr_classify</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move_Hive_Postgres</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move_Postgres_Hive</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w2v_train</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>w2v_vectorise</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grep</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Join</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Join4</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Left_Outer_Join</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keras_Projection</td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
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.

![Diagram of operator characteristics]

<table>
<thead>
<tr>
<th>Schemas</th>
<th>Filter</th>
<th>Projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>functionality</td>
<td>a</td>
<td>e</td>
</tr>
<tr>
<td>generated</td>
<td>ø</td>
<td>ø</td>
</tr>
<tr>
<td>projected-out</td>
<td>ø</td>
<td>c</td>
</tr>
<tr>
<td>input</td>
<td>a,b,c</td>
<td>a,b,c</td>
</tr>
<tr>
<td>output</td>
<td>a,b,c</td>
<td>a,b</td>
</tr>
</tbody>
</table>

Optimization via graph reconfiguration

Transitions generating equivalent workflow versions:

- **Swap** ($A$ and $B$)
- **Decompose** ($A$ and $B$)
- **Compose** ($A$ and $B$)
- **Factorize** ($A$ and $B$)
- **Distribute** ($A$ and $B$)
- **Filter** ($A$ and $B$)
- **Projection** ($A$ and $B$)
- **Join** ($A$ and $B$)
- **Calc** ($A$ and $B$)

![Diagram of optimization via graph reconfiguration]
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

<table>
<thead>
<tr>
<th>Applicability of transitions in based on the schemas</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>swap</strong></td>
</tr>
<tr>
<td>filter</td>
</tr>
<tr>
<td>calc</td>
</tr>
<tr>
<td>join</td>
</tr>
<tr>
<td>filter_calc</td>
</tr>
<tr>
<td>filter_join</td>
</tr>
</tbody>
</table>

Applicability table for swap and other operators
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

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
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

Verena Kantere
35
14.01.19
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`

Verena Kantere

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 with IRES

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:**
  - 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:**
  - 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:

<table>
<thead>
<tr>
<th>Workflow size</th>
<th>range</th>
<th>constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-200</td>
<td>20-50</td>
<td></td>
</tr>
</tbody>
</table>

- Workflow structure:
  - butterfly: 10-70% 25%
  - line: 10-70% 25%
  - fork: 10-70% 25%
  - tree: 10-70% 25%

- 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

![Experimental results for synthetic workflows](image)

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

Original workflows

Joint workflow
Multi-workflow optimization cont’d

The creation of a joint workflow $W_0$ of a set $W = \{W_1, \ldots, W_m\}$ 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
The processing cost of a joint workflow $W_o$ of workflows $W = \{W_1, \ldots, W_m\}$ with common parts $\{CP_1, \ldots, CP_n\}$ is:

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

where $l_i$ is the number of occurrences of common part $CP_i$ in $W$ and $sync_i$ is the cost of synchronization of execution of common parts.
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%].
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.

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

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.
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

![Diagram of dataflow and 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
A technique for automated re-calibration:

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

Recalibration process

1. Workflow augmented with recalibration points
2. Visualization of intermediate results
3. Workflow changed by the user
4. Execution has reached monitoring point
5. Preparation of a workflow part
6. Execution has reached if-then-else point
7. Execution has reached goto point
8. Materialize and execute
9. Get status
10. Workflow execution
Recalibration process

1. Workflow augmented with recalibration points
2. Visualization of intermediate results
3. Workflow changed by the user
4. Execution has reached monitoring point
5. Preparation of a workflow part
6. Execution has reached if-then-else point
7. Execution has reached goto point
8. Workflow changed by the user
9. Materialize and execute
10. Get status

Workflow execution

14.01.19
Verena Kantere
65

Recalibration process

1. Workflow augmented with recalibration points
2. Visualization of intermediate results
3. Workflow changed by the user
4. Execution has reached monitoring point
5. Preparation of a workflow part
6. Execution has reached if-then-else point
7. Execution has reached goto point
8. Workflow changed by the user
9. Materialize and execute
10. Get status

Workflow execution

14.01.19
Verena Kantere
66
Recalibration process

Workflow augmented with recalibration points
Visualization of intermediate results
Workflow changed by the user

Execution has reached monitoring point
Preparation of a workflow part
Execution has reached goto point

Workflow execution
Get status
Materialize and execute

Get status
Materialize and execute
Workflow execution

Recalibration process

Workflow augmented with recalibration points
Visualization of intermediate results
Workflow changed by the user

Execution has reached monitoring point
Preparation of a workflow part
Execution has reached goto point

Workflow execution
Get status
Materialize and execute
Monitoring intermediate results of ‘Peak Detection’ in PAW

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)
- **Apache Flink** (TU Berlin) (2014 – now)
- **Emma** (TU Berlin) (2015 – now)
- **BigDAWG Polystore System (UofC, MIT, Intel)** (2015 – now)
- **Rheem** (QCRI, HBKU) (2015 – now)

Comments and questions?

vkantere@uottawa.ca