

Online Tuning of Aggregation Tables for OLAP

Kai-Uwe Sattler
Katja Hose
Daniel Klan

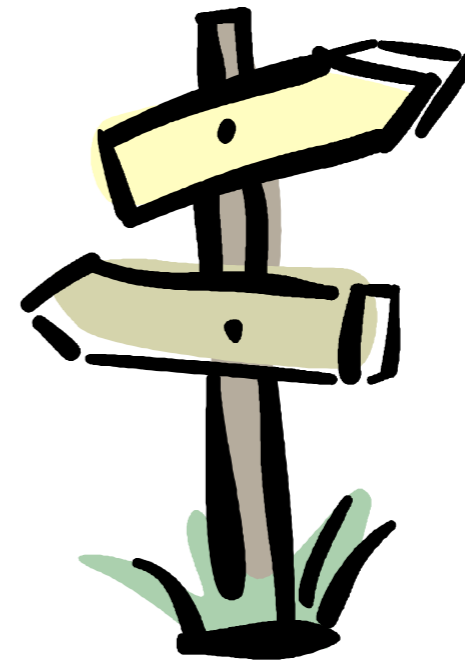
Database & Information Systems Group
Ilmenau University of Technology, Germany
www.tu-ilmenau.de/dbis



ILMENAU UNIVERSITY OF
TECHNOLOGY

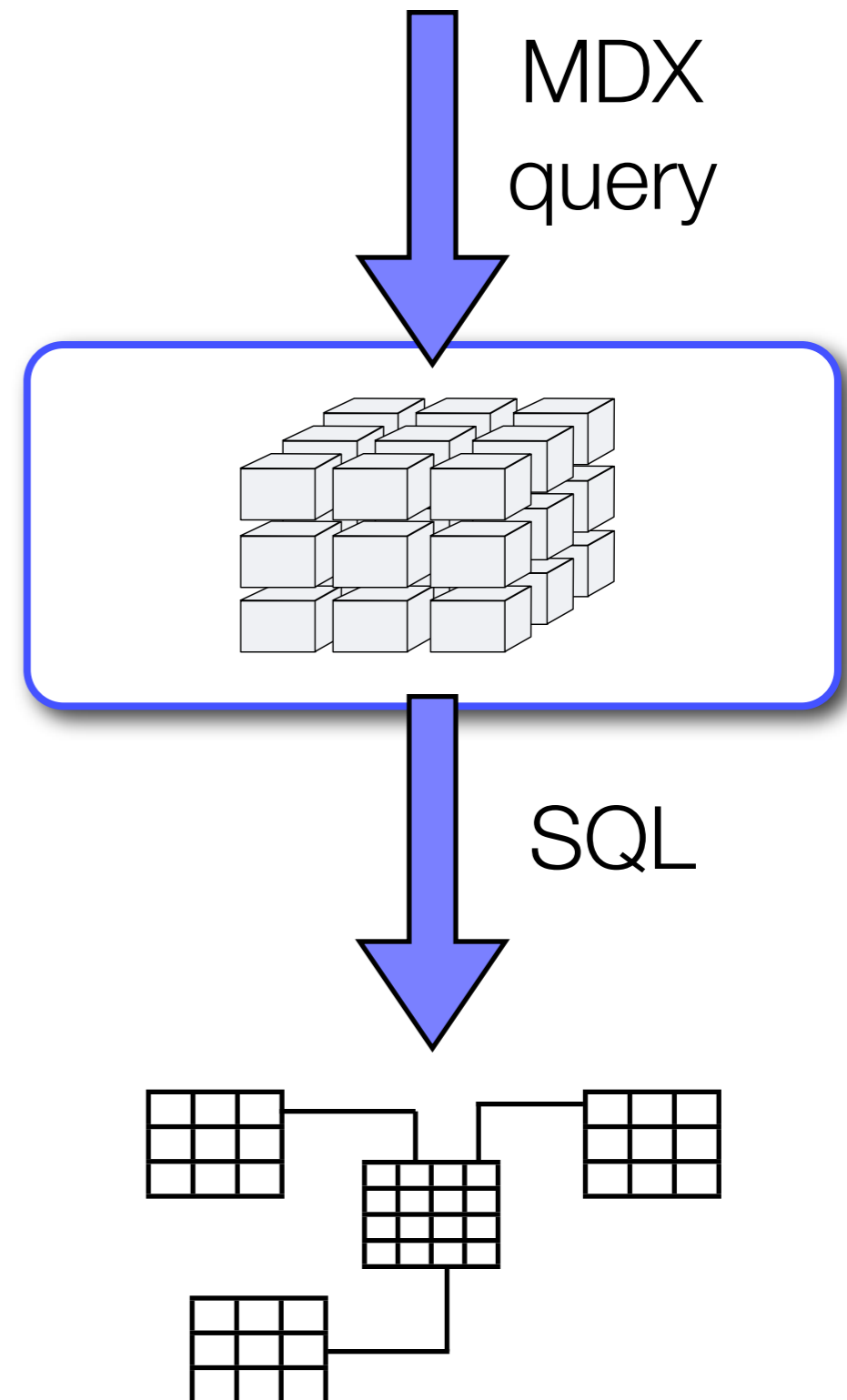
Outline

- Background & Problem Statement
- Foundations
- Finding the Optimal Configuration
- Implementation
- Evaluation Results
- Conclusion & Outlook



Background: OLAP Server

- middleware for processing analytical queries (e.g. MDX)
- uses a cube as logical data model
- mapped to a star schema
- multidimensional queries are translated into SQL
- examples:
 - OLAP engines from commercial DBMS vendors
 - Data Warehouse Accelerators
 - Open Source solutions like Mondrian



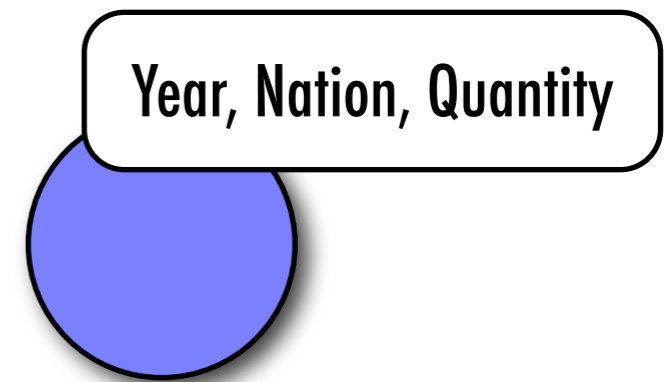
Problem Statement

- analytical queries are expensive
 - huge fact table, many joins, grouping/aggregation
- solutions: KIWI, indexing, [pre-aggregation](#), alternative backends
 - materialized views in relational DBMS
 - aggregation tables in Mondrian
- challenges
 - [view selection problem](#): which (intermediate) query results should be materialized? - huge search space, query containment problem, cost information about non-existing tables
 - [online tuning](#): adjust the set of aggregation tables - pre-defined workload for OLAP?

Foundations: From MDX to Aggregation Nodes

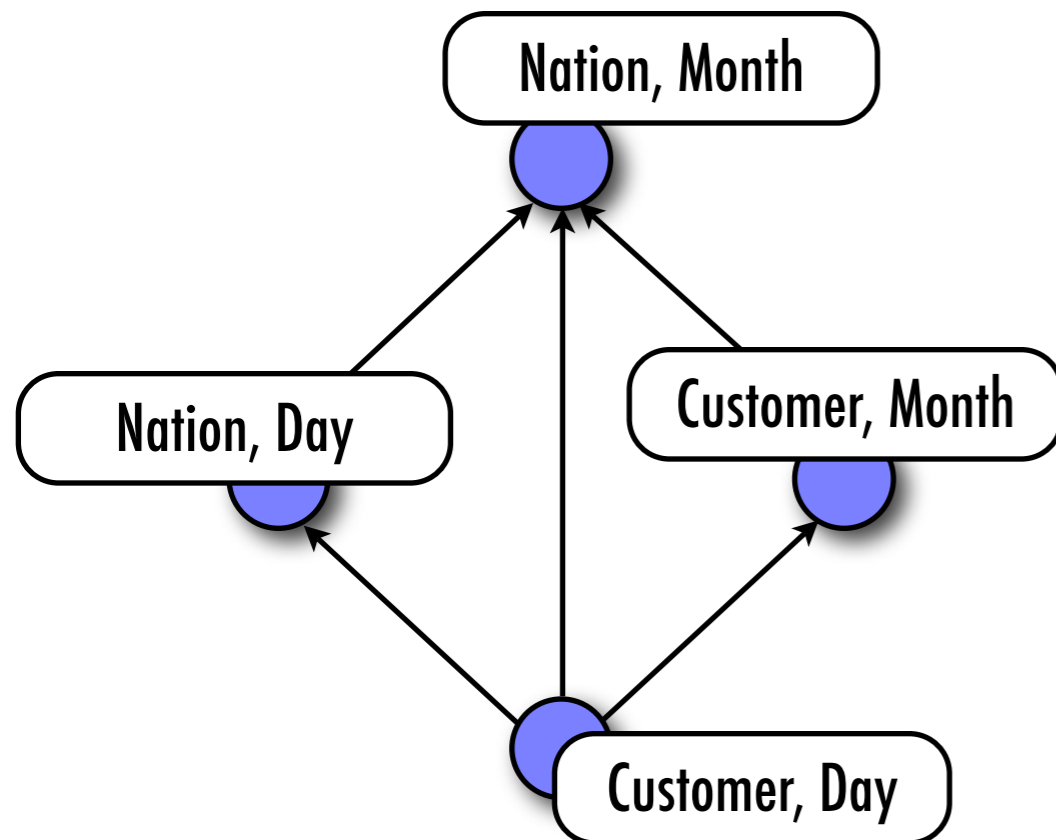
- a MDX query is represented by a (generalizing) aggregation node

```
SELECT Time.Year ON COLUMNS,  
Customer.Region.Nation ON ROWS  
FROM Sales  
WHERE (Measures.Quantity)
```



- derivability of aggregation nodes:
 - can be query q_i answered by node v_j ?
 - at which costs (= benefit of v_j wrt. q_i)

Aggregation Lattice



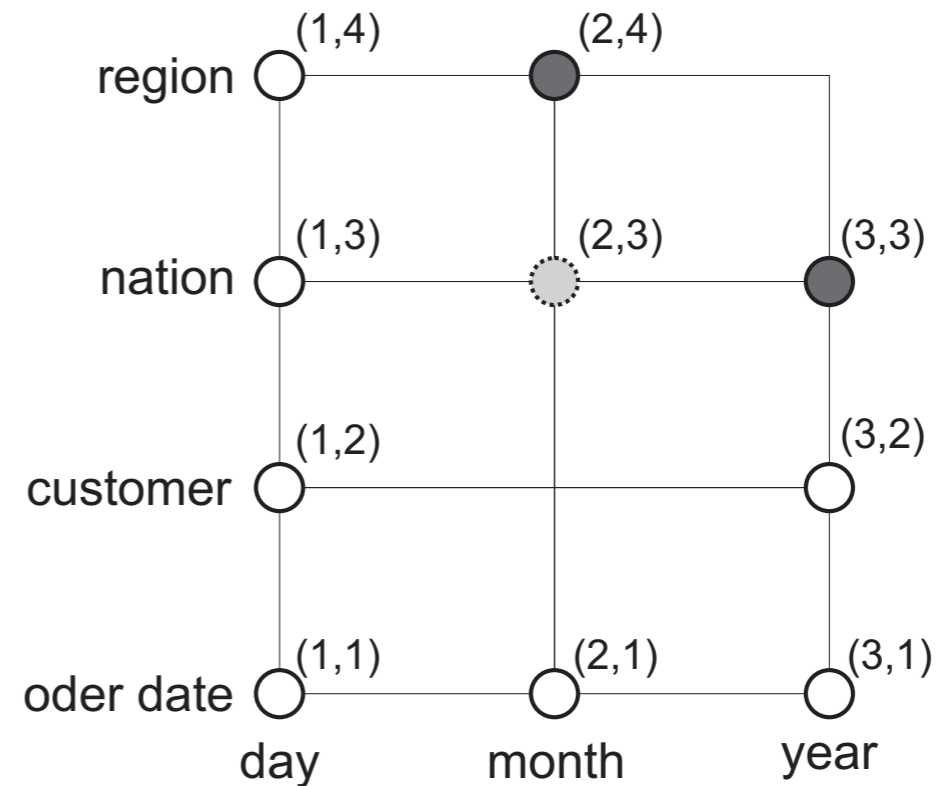
- number of nodes

$$\#nodes = m \cdot \prod_{i=1}^d l_i$$

- collect only **relevant** nodes
 - nodes corresponding to queries
 - common base nodes for relevant nodes
- **efficient strategy** for detecting derivability

Aggregation Lattice

- in each dimension: levels are represented by numbers
- aggregation node: described by coordinates



- derivability check: v_2 is derivable from v_1

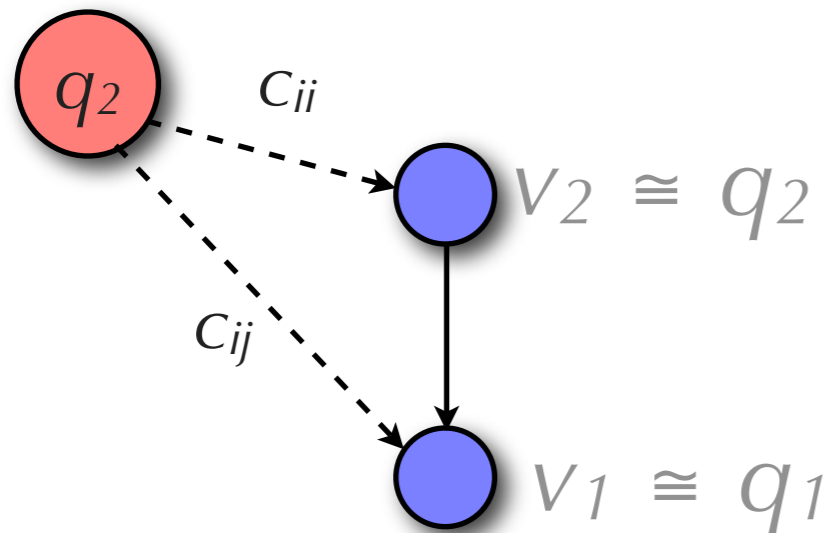
$$\forall i \in D : v_1[i] \leq v_2[i]$$

- greatest common base node for a and b

$$\forall i \in D : v[i] = \min\{a[i], b[i]\}$$

Cost Model

- estimating query processing costs in order to predict the benefit of a configuration



- C_{ii} ... full table scan
- estimate cardinality of v_i
- C_{ij} ($i \neq j$) ... typical query plan

GroupAggr
+---Sort
+---SeqScan(v_j)

- estimate cardinality of v_j = number of distinct value combinations
- COLSCARD (CIKM'05): based on Kronecker product of individual frequency vectors

Finding the Optimal Configuration: Use Cases

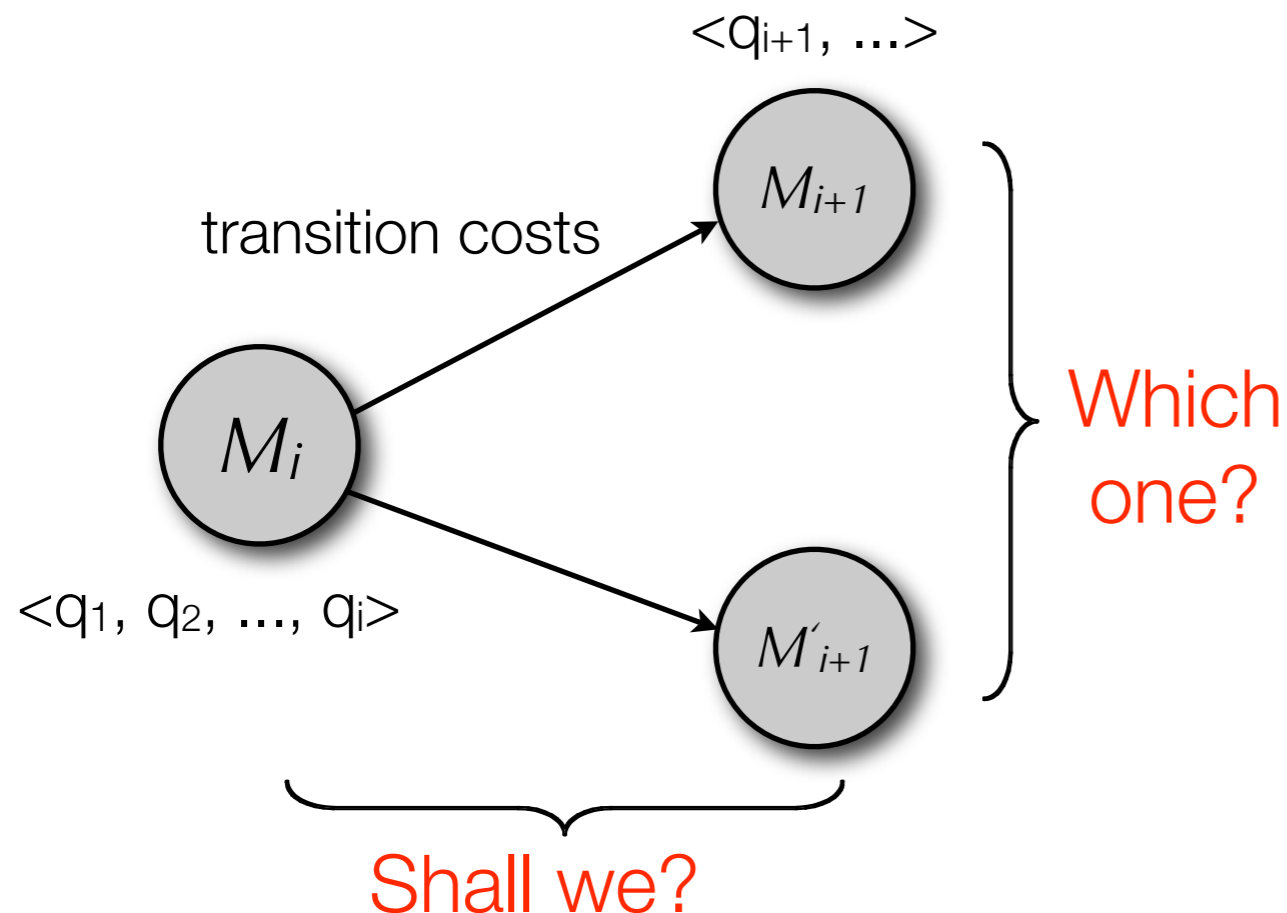
- Goal: maintain an optimal configuration = set of aggregation nodes that
 1. minimizes query execution costs for a given workload
 2. wrt. given size constraints
- usage:
 - **static**: optimization problem for a fixed workload (knapsack problem)
 - **alert**: check periodically if there is a better configuration, e.g. in case of workload drifts
 - **online**: adjust the configuration continuously

The Online Optimization Problem

- ice cream vendor problem: two flavors
 - vanilla \$1 by machine, \$2 by hand
 - chocolate \$2 by machine, \$4 by hand
 - changing the state of the machine: \$1



use the machine,
change the state, or
make by hand?



- metrical task systems
 - symmetric transition costs
 - limited number of states
 - randomized algorithms

Algorithm

- heuristic-based choice of transition: cumulative penalty of remaining a configuration M_{i-1} instead of transitioning to M_i

- penalty for q_i $\Delta_i = cost(q_i, M_{i-1}) - cost(q_i, M_i)$

- cumulative penalty $penalty(M_i) = penalty(M_{i-1}) + \Delta_i$

Input: incoming query q_j , configuration M_{i-1}

$$M_i = M_{i-1} \cup \{v_j\}$$

$$\Delta_i = cost(q_j, M_{i-1}) - cost(q_j, M_i)$$

$$penalty(M_{i-1}) = penalty(M_{i-1}) + \Delta_i$$

if $penalty(M_{i-1}) > T$ then

$$M_j = \text{best-neighbor-config}(M_{i-1})$$

materialize(M_j)

$$T = \text{transition}(M_{i-1}, M_j); \text{penalty}(M_j) = 0$$

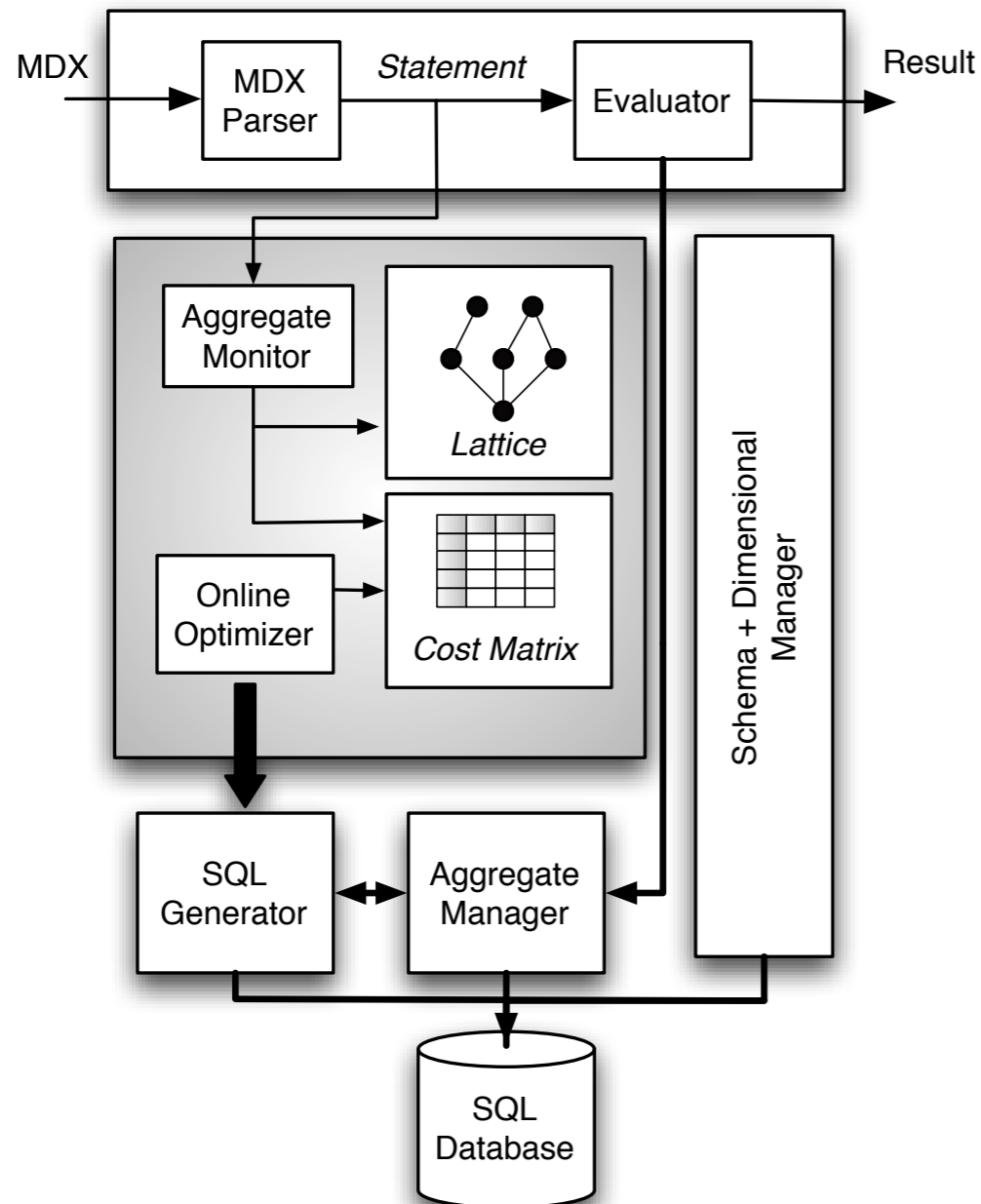
Indicator for
necessary transition

Find best
configuration wrt. the
previous k queries

perform transition

Prototype Implementation

- implemented as extension to Mondrian (demo @VLDB'08)



GraphicalLatticeViewer

File Edit Settings

Lattice Structure

Node Details

Name: 12
DerivedNodes: 7
DerivableNodes: 2

Statistics

Node Call Frequency: 1
AggTable Materialized: true
AggTable Cardinality: 1325
Used Space(KB): 164.0

Dimensions:

[Measures].[Store Sales]
[Product].[Product Family].[Product Department].[Product Subcategory].[Product Category].[Product Class]
[Store].[Store Country].[Store State].[Store City]
[Time].[Year]

Optimization

AggTable	14	15	16	17	
Base	30	1446.51	1390.36	1531.93	150
1	4	840.21	834.88	850.86	845
2	1	776.49	771.16		781
3	5	188.95	187.76		
4		60.03	59.68		
5			59.44		
6	3	257.96	256.28	261.30	259
7	2	234.15	232.48		235
8	1	51.97	51.63		
9		114.71	114.61		
10			107.04		
11	4	51.25	50.96	51.83	51
12	0	51.00	50.71		51
13	7	68.79	68.74		
14		22.76	22.90		
15			22.76		
16				166.83	166
17					151
18					

Alerter

100%
optimized config
realized config

Materialize Alerter Config
Find Optimal Config

SQL Statement:

```

SELECT
  "time_by_day"."the_year" AS "the_year",
  "product_class"."product_department" AS "pr
  "product_class"."product_family" AS "product
  "product_class"."product_subcategory" AS "pr
  "product_class"."product_category" AS "produ
  "store"."store_state" AS "store_state",
  "store"."store_country" AS "store_country",
  "store"."store_city" AS "store_city",
  sum("sales_fact_1997"."store_sales") AS "sto
  COUNT(*) AS "fact_count"
FROM
  "sales_fact_1997" "sales_fact_1997",
  "time_by_day" AS "time_by_day",
  "product_class" AS "product_class",
  "product" AS "product",
  "store" AS "store"
  
```

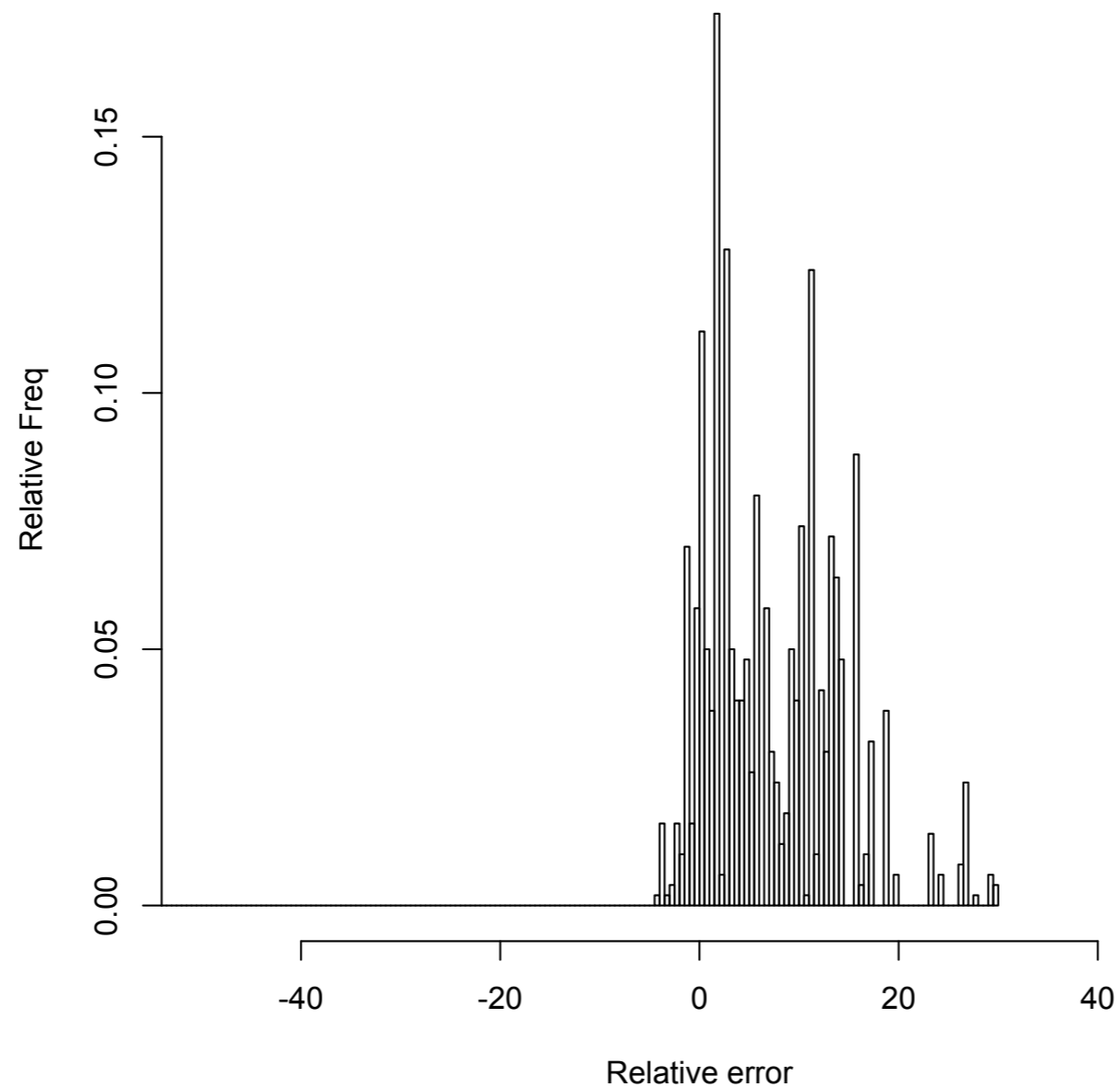
Create/Drop AggTable
Run Explain Query

Evaluation: Setup

- backend: PostgreSQL 8.3 on Linux, TPC-H dataset, SF=1 GB
- workload: 1000 randomly chosen queries/nodes
 - different shift strategies: random, repetition, random shift between two query sets, dimensional shift (between different dimensions)
- strategies: online vs.
 - no opt. = aggregation tables
 - greedy = greedy strategy based on all already seen queries
 - exhaustive = consider all possible configurations for all already seen queries
- experiments:
 - cumulative query execution costs
 - cumulative transition costs

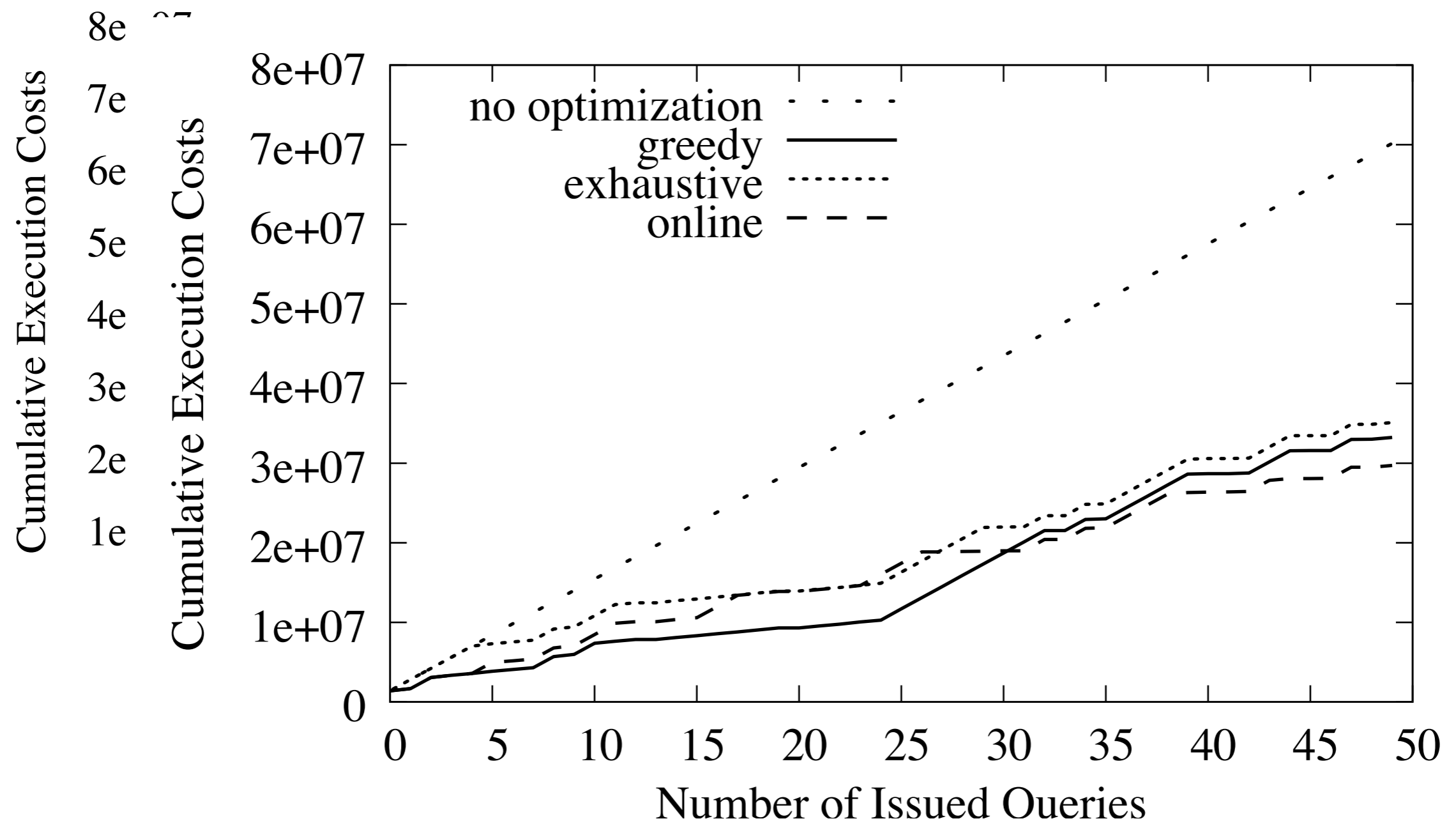
Evaluation: Estimation Error

- estimated vs. actual executions costs of 1000 randomly chosen query nodes



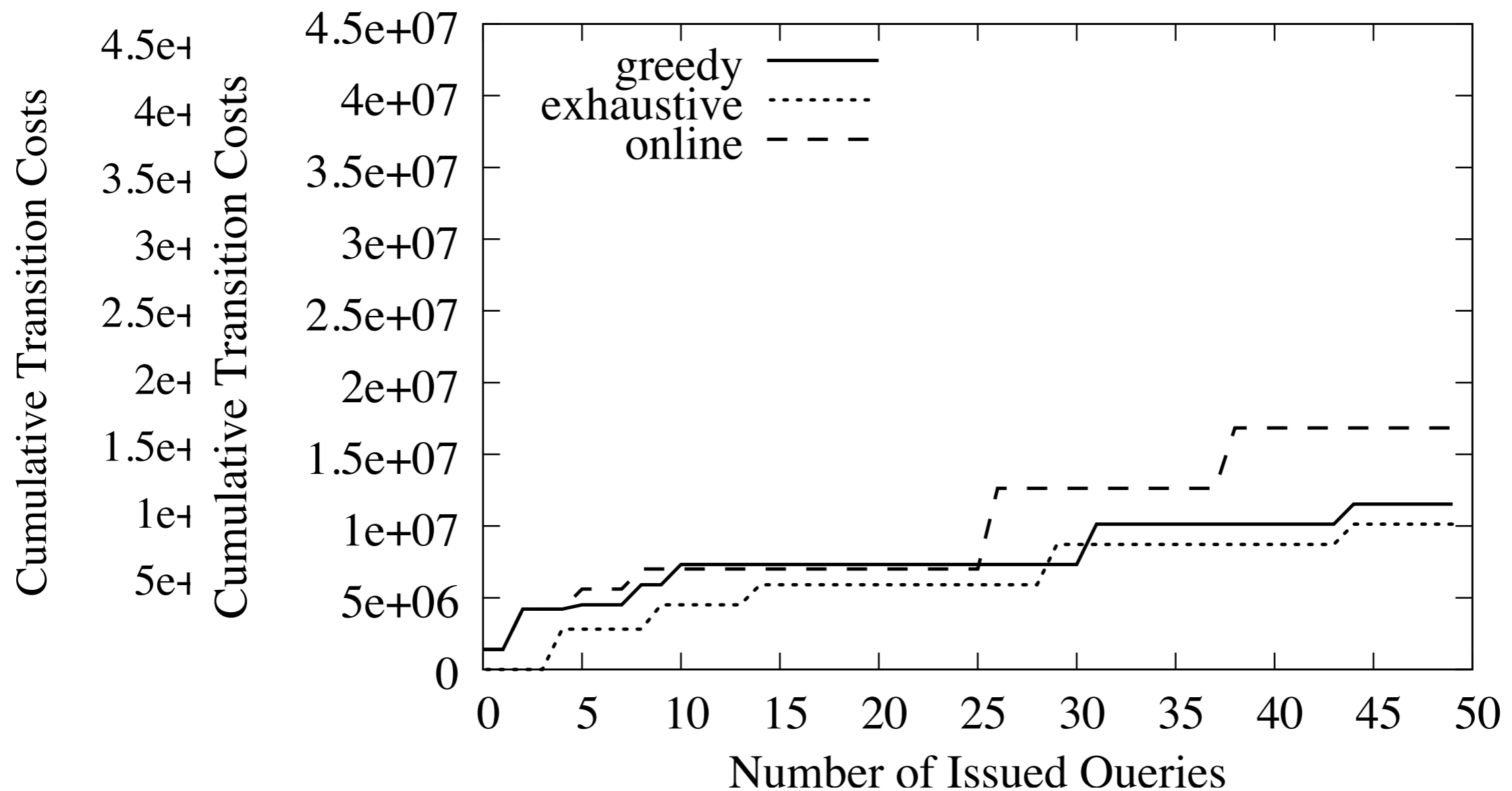
Evaluation: Execution Costs

- Random vs. Shift



Evaluation: Transition Costs

- Random vs. Shift



Conclusion & Outlook

- online approach for maintaining an optimal configuration of aggregation nodes
 - maintain derivability relationships in a lattice with coordinates
 - collect & estimate costs
 - online algorithm
- outlook
 - investigate other optimization approaches (game theory & economic models, ...)
 - better cost estimations
 - consider additional physical structures and alternative backends

