Automatic Column-Based Allocation in Distributed Data Warehouses

Tilmann Rabl
Chair of Distributed Information Systems
University of Passau
10.12.2008
Agenda

- Introduction
- Allocation
  - Complete approach
  - Column based
- Evaluation
- Conclusion
Introduction

- **Database cluster**
  - Shared nothing
  - 3-Tier architecture
- **Automatic allocation**
  - Automatic distribution/replication of data
  - Local query processing
  - Basing on query history
  - Periodical adaption at runtime
  - Inherent load balancing
- **Self scaling**
  - Number of backends depends on workload
Basic Idea
Dynamic Allocation

• Goals
  ▫ Efficient use of partial replication
  ▫ Balance workload
  ▫ Reduce disk usage / replication
  ▫ Adopt to changing request profiles / cluster sizes

• Approach
  ▫ Classify requests
  ▫ Create request profile
  ▫ Find good allocation
  ▫ Implement allocation
Classification

• Based on tables/column
• Characteristics sets not disjoint

• Determine fraction of overall workload
• Cost functions:
  □ Number of requests
  □ Sum of execution times
  □ Cost estimation of the query optimizer
Allocation Algorithm

• Process all statements locally
• Partial replication
• Optional
  ▫ Column-based
  ▫ K-safety

• Different versions
  ▫ Linear programming
    • optimal solution – slow
  ▫ Greedy heuristic
    • good performance – needs improvement
Greedy Allocation Algorithm

Sort request classes descending
Foreach class {
    Foreach backend {
        determine similarity of content with class
    }
    While class is not fully distributed {
        place class at backend with highest similarity
    }
}
Improving Implementing Allocations

• **Meta heuristic improvement**
  ▫ Simulated annealing / evolutionary strategy
  ▫ Improving replication / disc usage ratio

• **Implementing allocations**
  ▫ **Matching Allocations**
  ▫ **Hungarian Method:**
    • Complete bipartite graph
    • Perfect Matching in $O(n^3)$
Column-Based Allocation

- Vertical partitioning
- Finer classification
- Less disk usage
- Less I/O
Column-Based - Example

- Sales statistics for products 50%
  FactTable, Product, Date
- Sales of a salesperson 25%
  FactTable, Salesperson, Date
- Regional bestsellers 25%
  FactTable, City, Product
Column-based - Example II

- Referenced columns
Column-Based - Example III

• 2 backends
  ▫ 30% saved

• 4 backends
  ▫ 50% saved
TPC-H Benchmark

- DSS/OLAP benchmark
- 22 queries
- Read only
- Scale 0.1
  - Derby DB
- Mainly CPU
Disk Usage

- einzelne Datenbank
- spaltenbasierte Anfragesklassen
- tabellenbasierte Anfragesklassen
- totale Replikation

Replikationsgrad vs. Backends
Throughput

- + totale Replikation
- × Allokationsheuristik

Durchsatz (Anfragen/sek)

Backends
Bad Throughput for 5 Backends
Adoption of the Allocation
Conclusion

• Scales good for read dominant workload
• Good reduction of disk usage

• But what about,
  ▫ Varying query patterns?
  ▫ Heterogeneous environments?
  ▫ OLTP?
Current and Future Work

• Horizontal partitioning
• Distributed joins
• Classification based on cost model
• Support for heterogeneous backends
• Time series analysis of log data
  ▫ Estimation of future query profiles
  ▫ Workload estimation
Questions?