Dynamic Allocation in a Self-Scaling Cluster Database

Tilmann Rabl

University of Passau

VLDB DMG, September 23th 2007
Introduction

Self-Scaling Cluster

Dynamic Allocation

Implementation

Benchmarks

Conclusion

Tilmann Rabl

University of Passau

Dynamic Allocation in a Self-Scaling Cluster Database
Introduction

Cluster Database

- COTS hardware
- Minimal requirements
Introduction

Cluster Database
- COTS hardware
- Minimal requirements

Dynamic Allocation
- Periodical analysis
- Adaption at runtime
- Inherent loadbalancing
Introduction

Cluster Database
- COTS hardware
- Minimal requirements

Dynamic Allocation
- Periodical analysis
- Adaption at runtime
- Inherent loadbalancing

Self Scaling
- Number of backends depends on workload
- New backends need no installation
RAIDb

- Redundant Array of Inexpensive Databases
- Shared nothing architecture
- Read Once / Write All Available
- RAIDb levels
  0  Partitioning
  1  Full Replication
  2  Partial Replication
RAIDb

- Redundant Array of Inexpensive Databases
- Shared nothing architecture
- Read Once / Write All Available
- RAIDb levels
  0  Partitioning
  1  Full Replication
  2  Partial Replication
RAIDb

- Redundant Array of Inexpensive Databases
- Shared nothing architecture
- Read Once / Write All Available
- RAIDb levels
  0 Partitioning
  1 Full Replication
  2 Partial Replication
RAIDb

- Redundant Array of Inexpensive Databases
- Shared nothing architecture
- Read Once / Write All Available
- RAIDb levels
  0 Partitioning
  1 Full Replication
  2 Partial Replication
Example

Database with 3 relations, 4 query classes

A 30%
B 25%
C 25%
AB 20%
Example

Database with 3 relations, 4 query classes

```
A  B  C
A  B
B  C
A B
```

A  30%
B  25%
C  25%
A B  20%
Example

Database with 3 relations, 4 query classes
Self Scaling Cluster

- Size of the cluster adopts to database workload
- Backends and controller are monitored
- Measurements:
  - Query queue
  - Processor load
  - Disk space
  - Free memory
- Scoring model determines if cluster is scaled
Scaling Procedure

- **Up scaling**
  - New backend is chosen
  - Database software is transmitted via SFTP
  - Database is started via SSH
  - Database is integrated in cluster
  - Data is transmitted

- **Down scaling**
  - Data is transmitted from backend
  - Data is removed
  - Database is shut down
  - Connection to backend is closed
Scaling Procedure

- **Up scaling**
  - New backend is chosen
  - Database software is transmitted via SFTP
  - Database is started via SSH
  - Database is integrated in cluster
  - Data is transmitted

- **Down scaling**
  - Data is transmitted from
  - Data is removed
  - Database is shut down
  - Connection to backend is closed
Dynamic Allocation

Goals

- Efficient use of partial replication
- Balance workload of (read) requests
- Reduce disk usage / replication
- Adopt to changing request profiles / cluster sizes
Dynamic Allocation

- **Goals**
  - Efficient use of partial replication
  - Balance workload of (read) requests
  - Reduce disk usage / replication
  - Adopt to changing request profiles / cluster sizes

- **Approach**
  - Classify requests
  - Create request profile
  - Find good allocation
  - Implement allocation
Classification of Requests

Classification

- Referenced relations classify request
  - select * from A
  - select x from A
  - Class A

- Relation sets are not disjunct
  - select * from B
  - Class B
Classification of Requests

Classification

- Referenced relations classify request
  - select * from A
  - select x from A
- Relation sets are not disjunct
  - select * from B

Request profile

- Foreach class determine fraction of overall workload
- Cost functions
  - Number of requests
  - Sum of execution times
  - Cost estimation of the query optimizer
Finding an Initial Allocation

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
    }
}
Finding an Initial Allocation

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
    }
}

Example:

```
Class A
Class B
Class AB
```
Finding an Initial Allocation

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
    }
}

Example:

```
Class A
Class B
Class AB
```

```
Class B
Class B
```
Finding an Initial Allocation

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
    }
}

Example:
Finding an Initial Allocation

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
    }
}

Example:
Improving and Implementing the Allocation

Simulated Annealing

- Metaheuristic
- Neighborhood relation:
  - Swap one request class
Improving and Implementing the Allocation

Improving the Allocation

Simulated Annealing
- Metaheuristic
- Neighborhood relation:
  - Swap one request class

Implementing the Allocation

- Hungarian Method
- Cost minimal matching ($O(n^3)$)
Sequoia Architecture

- Formerly known as C-JDBC
- RAIDb implementation
- Java based
- Works with arbitrary backends
SCMT Architecture

- Java 1.6
- Uses JMX to interact with Sequoia
- Stores History in HSQLDB
- Apache Derby for backends
TPC-H

- Decision Support benchmark
- Complex queries

![Graph showing throughput vs. number of backends]

- SingleDB
- RAIDb-1
- RAIDb-2

Throughput
TPC-H

- Decision Support benchmark
- Complex queries

Number of Replications

Number of backends
TPC-H

- Decision Support benchmark
- Complex queries

![Graph showing disk usage vs. number of backends for different configurations of TPC-H benchmark](image)

**Title:** Dynamic Allocation in a Self-Scaling Cluster Database

**Authors:** Tilmann Rabl

**Institution:** University of Passau
TPC-W

eCommerce benchmark
Transaction oriented

Throughput

Throughput Read-Only

Tilmann Rabl
Dynamic Allocation in a Self-Scaling Cluster Database
TPC-W

- eComerce benchmark
- Transaction oriented

Disk Usage

Disk Usage Read-Only

Tilmann Rabl

Dynamic Allocation in a Self-Scaling Cluster Database
Conclusion

Results

- Scales good for read dominant workload
- Inherent problems with write requests
- Good reduction of disk usage
Conclusion

Results

- Scales good for read dominant workload
- Inherent problems with write requests
- Good reduction of disk usage

Future Work

- Support for heterogeneous clusters
- Time based analysis of query history
- Allocation of fragments
Questions?

Thank you for your attention!