Seminar Selftuning Databases
Data-Placement

Mathias Körbs

13. Januar 2004

Content

- Introduction
- Data-Placement
- The TS-Greedy algorithm
- Conclusion

Motivation

- Storage of data should be as efficient as possible
- Where to store which database-object? → Data Placement
- Data-maintenance should be as easy as possible
- Database-administrator (DBA) has to take care of everything → Self-Tuning

Data Placement

What does Data Placement mean?

- Distribution of database-objects to all available physical media
- Database-objects are tables, indexes and materialized views
- Physical media are harddisks, raid-arrays but also nodes in a parallel database-environment
- Database-performance and serviceability are central requirements

Self Tuning

- Optimal tuning of a database-system is very expensive
- Requirements change during lifecylce of the DBS (increasing accesses, changing access-pattern)
- DB-Adminstrator has to permanently adjust DBS to the changing requirements → expensive
- Goal is to reduce the cost
- DBMS ought to tune itself as much as possible

Storage of DB-Objects

- A DB-object is stored in *only one* tablespace
- Tablespaces are "containers", that might contain more than one DB-object
- Tablespaces can be
  - Files in the filesystem of the operating-system
  - Partitions on disks
  - Many partitions on many disks

Distribution of DB-OBjekts

- Simpliest case: all DB-objects are on one disk
  - Easy to maintain
  - Objects accessed by a query are read sequential
- Distributing each object uniformly to all disks (full-striping)
  - Easy to maintain
  - Again, data is read sequential
  - Maximum throughput
- Until now, we didn't consider the disk's access-time

Influence of the Access-Time

- Using full-striping doesn't reduce the access-time
- Example: Merge-Join of two huge tables
  - Seek, Read R_1, Seek, Read R_2, Seek, Read R_1, Seek, Read R_2, ...
  - Time for reading decreases with each additional disk
  - Time for Seek doesn't change (the relative part on the whole operation increases)

Effective Placement

- DB-Objects, that are co-accessed are divided to different sets of disks
- Seek's on distributed objects can be accessed parallel
- But fewer parallelism and throughput
- Trade-off between high throughput and short access-time has to be found
- Higher effort for database-administrator
- → starting-point for Self-Tuning

Starting-Point of the AutoAdmin-Project

- Goal: automating the search for an effective database-layout
- DB-Objects should be distributed to available disk-drives
- Trade-off beetween high throughput and short access-time $\rightarrow$ formulated as an optimzation-problem

Available Information

- Database
  - Tables and their sizes
  - Indexes
  - Materialized views
- Workload
  - All queries, that are executed at the database
  - Weight of each query compared to the other queries in the workload
- Disks
  - Number of disks available
  - Accesstime, read-, writerate and capacity of each disk
  - Availability-properties (None, Parity, Mirroring)

The TS-Greedy-Algorithm

- Creating the access-graph from workload
- Partition of the access-graph, the sum of weights of the interpartition-edges has to be the maximum
- Distribution of the partitions to disk
- Distributing all disks, left from the previous step, to increase parallelism

Creating the Access-Graph

- The workload is displayed as a graph
- Analysis of the execution-plans of the workload's queries (received from the optimizer)
- Each DB-object is represented by a node
- Weight of a node represents the number of accessed blocks
- Edge beetween two nodes exists when these nodes are co-accessed by one or more queries
- Weight of an edge represents the sum of all blocks, that are co-accessed, blocks that are co-accessed in more than one query are counted more than once

Partition of the Access-Graph

- Access-Graph has to be partitioned so that the sum of all cutted edges is maximized
- Number of partitions has to be equal or smaller than the number of available disks
- Ideally, no partition contains co-accessed objects
- Project uses Kernighan-Lin-Algorithm to partition the graph

Distribution of the Partitions to Disk

- Disks are sorted in descending order by transferrate
- For each partition (in order of descending nodeweight-sum)
  - Assign the partition's DB-Objects (using full-striping) to the smallest set of disks where they fit
  - Starting at the disk with the highest transferrate
- Already assigned disks can not be distributed anymore
- If a partitionP_1 doesn't fit on any unassigned set of disks
  - Get an already assigned partition P_2 so that the sum of edgeweights beetween P_1 and P_2 is minimal and assign P_1 to the same set of disks as P_2

Distribution of the remaining Disks

- Usually, after the previous step some disk-drives are unassigned
- Execute on the current DB-layout L:
  - Create alternatives of L, by assigning k unassigned disks to each object
  - Get a layout-alternative L' with smallest C
  - If C' of L' smaller than C of L reread this slide with L' as new L
- $C = \Sigma w_q \cdot cost(Q, L)$
- $cost(Q, L)$ estimates the io-responsetime of the query Q (contains seek- and transfertime, but no cpu-time)

Comments

- First TS-Greedy minimizes the co-accesses on DB-objects
- And increases parallelism in a second step
- Parameter $k = 1$ delivers good results
- Complexity: $O(m^{k+1}n^2 + n^2 \cdot log(n))$; m number of available disks, n number of nodes in access-graph

Validation of the Cost-Model

Query	Δt_{exec}	Δt_{est}
Query 3	44%	54%
Query 9	30%	40%
Query 10	36%	51%
Query 12	32%	55%
Query 18	16%	31%
Query 21	40%	9%
TPCH-22	25%	20%

- Δt_{exec} measured improvement of the runningtime
- Δt_{est} expected improvement of the runningtime (expectations based on cost-model)

Effectiveness of the Algorithm

- Improvement of the runningtime compared to full-striping

Workload	Δt_{est}
WK-CTRL1	ca. 25%
WK-CTRL2	ca. 50%
TPCH-22	ca. 20%
SALES-45	ca. 40%
APB-800	0%

- Comments to the workloads in [ACDN03]
- Algorithm scales as expected $O(m^{k+1}n^2 + n^2 \cdot log(n)))$

Conclusion

- An effective database-layout is very expensive
- Solution: automated generation of the DB-layout
- Automated generation of the DB-layout delivers better results than full-striping
- Currently no commercial implementation is available

Thank You

- for your attention

Literatur

[ACDN03] S. Agrawal, S. Chaudhuri, A. Das, and V. Narasayya. Automating layout of relational databases. *ICDE*, 2003.