Index Selection tools in Microsoft SQL Server and IBM DB2

Seminar: Self-Tuning Databases

Marcel Giard

26.11.2003

Structure

- Introduction
- Microsoft SQL Server Index Selection Tool
- IBM DB2 Advisor
- Conclusion

Introduction

- Indexes have great impact on performance of a DB
- configuration: set of indexes
- goal: support DBA with powerfully tool
- older approaches independent DB optimizer
- using the DB optimizer avoids asynchrony

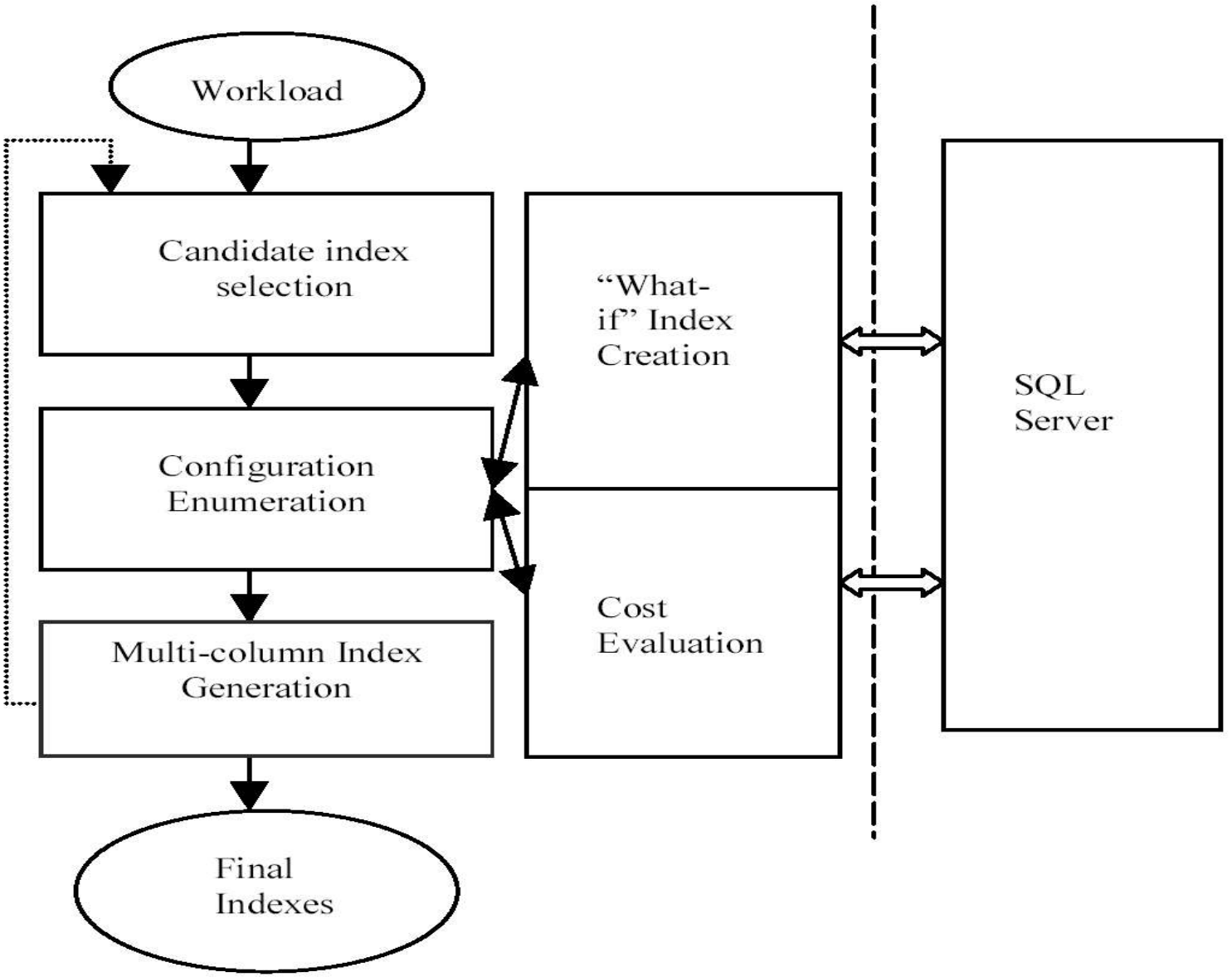
Problem

- Ideal: pick optimal index configuration from all possible configurations for a given workload
- for a table with n columns: $\sum_{k=1}^{n} n!/(n-k)!$
- e.g. a table with 10 has 6235301 possible indexes
- not counting different types of indexes
- analogue for index configurations for a DB with n tables

Microsoft Index Selection Tool

- part of Auto Admin
- time to perform a workload represents the cost of the workload
- uses own optimizer for cost estimation
- different algorithms to narrow the search room
- iterative (first single column indexes then multi column indexes)

Architecture



Candidate Index Selection

- selects promising index configurations
- breaks down a workload of n queries in n workloads of one query
- Index selection for each of them (seed: indexable columns)
- union of results builds set of candidate index configurations

Configuration Enumeration

- number of all subsets of the candidate indexes still too big
- uses a greedy algorithm
- start with a pair of indexes and add successive the one with best benefit-cost ratio

Cost Evaluation

- lowers number of optimizer calls
- keeps track of former cost estimations of the optimizer and tries to derive costs for some queries from it
- e.g.: if for one query in the workload the new configuration only differs in indexes not used in this query the old cost can be used

“What-if” Tool

- uses modified SQL Server to operate with hypothetical indexes
- generating statistics for these indexes
- changing catalogue tables
- call optimizer in “no-execute” mode
- returns costs
- undoes changes

Multi-column Index Generation

- iterative approach
- create multi-column indexes with increasing width
- builds new multi-column indexes by adding a indexable column to an existing index
- generates new configurations by adding multi-column indexes to existing configurations
- gives them to the Candidate Index Selection

Test Results

Workload	Number of Joins	% Pure Queries	% Update Stmts.	# admissible Indexes
TPCD_1	0, 1, 2	70	30	75
TPCD_2	3, 4, 5	70	30	143
TPCD_3	0-5	100	0	123
TPCD_4	0-5	50	50	108
TPCD_O	-	100	0	124

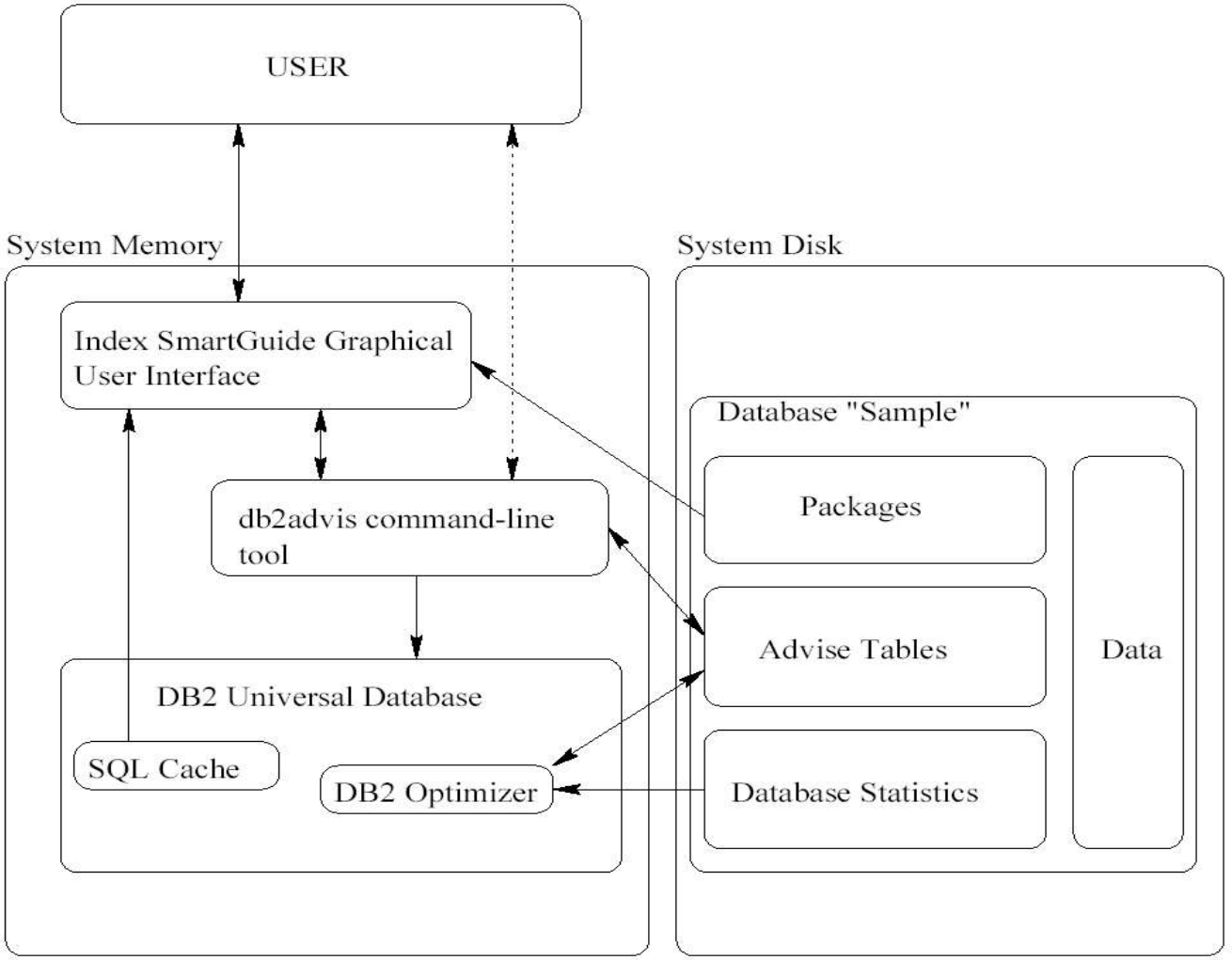
numbers from Microsoft

	Num. Candidate indexes	Num. optimizer calls	Num. Configs. Enum.	Running time	Drop in quality
TPCD_1	38%	52%	51%	24%	0%
TPCD_2	33%	8%	28%	12%	7%
TPCD_3	43%	11%	57%	14%	6%
TPCD_4	32%	11%	39%	18%	0%
TPCD_O	30%	3%	33%	10%	1%

IBM DB2 Advisor

- let the optimizer recommend candidate indexes
- each query in the workload is only called once
- all in one call
- index selection as variant of the Knapsack Problem
- input: workload + statistics of the DB
- output: recommended indexes

Architecture



Single Query Optimization Idea

- extension of existing optimization process
- idea:
  - suppose all possible indexes available
  - run optimization
  - indexes chosen by the optimizer are the optimal
- 2 ways in real world:
  - “Brute Force and Ignorance” (BFI)
  - “Smart column Enumeration for Index Scans” (SAEFIS)

Single Query Optimization Algorithm

ALGORITHM 1:

RECOMMEND INDEXES(Statement S)

1. Enable ”RECOMMEND INDEXES” mode
2. Enter the DB2 Optimizer
3. Inject the schema with virtual indexes using SAEFIS and generate their statistics
4. Inject the schema with virtual indexes using BFI and generate their statistics
5. Construct the best plan for S by calling the DB2 Optimizer
6. Scan the optimal plan, searching for virtual indexes
7. Submit these indexes back to the user as ”recommended”.

Workload Optimization Idea

- Knapsack Problem approach
- indexes selected in order of their benefit up to a maximal size
- then create variants by swapping small sets of indexes and test them
- if its cheaper its the new solution
- duration determines the quality of the solution

Workload Optimization Algorithm 1

ALGORITHM 2:

1. Get Workload W, including the frequency of execution

of each statement.

2. R = ~ 3. For each Statement S in W,

(a) EXPLAIN S with existing indexes, returning

S.cost with existing indexes.

4. For each Statement S in W,

(a) EXPLAIN S in RECOMMEND INDEX mode,

i.e. with virtual indexes

(b) R = R RECOMMEND INDEXES(S)

Workload Optimization Algorithm 2

ALGORITHM 2:

5. For each index I in R

(a) I.benefit = S.cost with existing indexes -

S.cost with virtual indexes

(b) I.size = bytes in index

6. Sort indexes in R by decreasing benefit-to-cost ratio.

7. Combine any index subsumed by an index with a

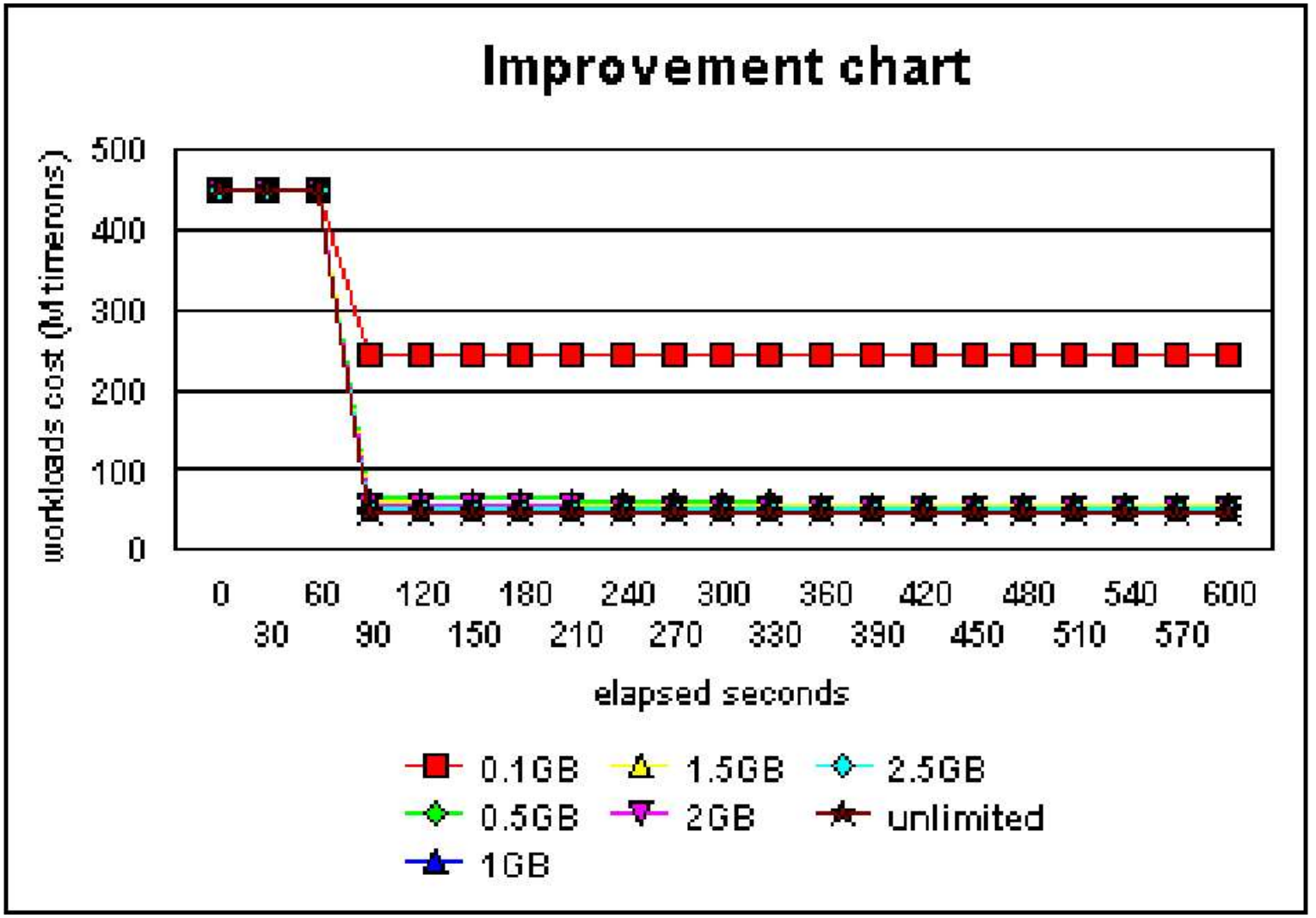
higher ratio with that index.

8. Accept indexes from set R until disk constraint is exhausted.

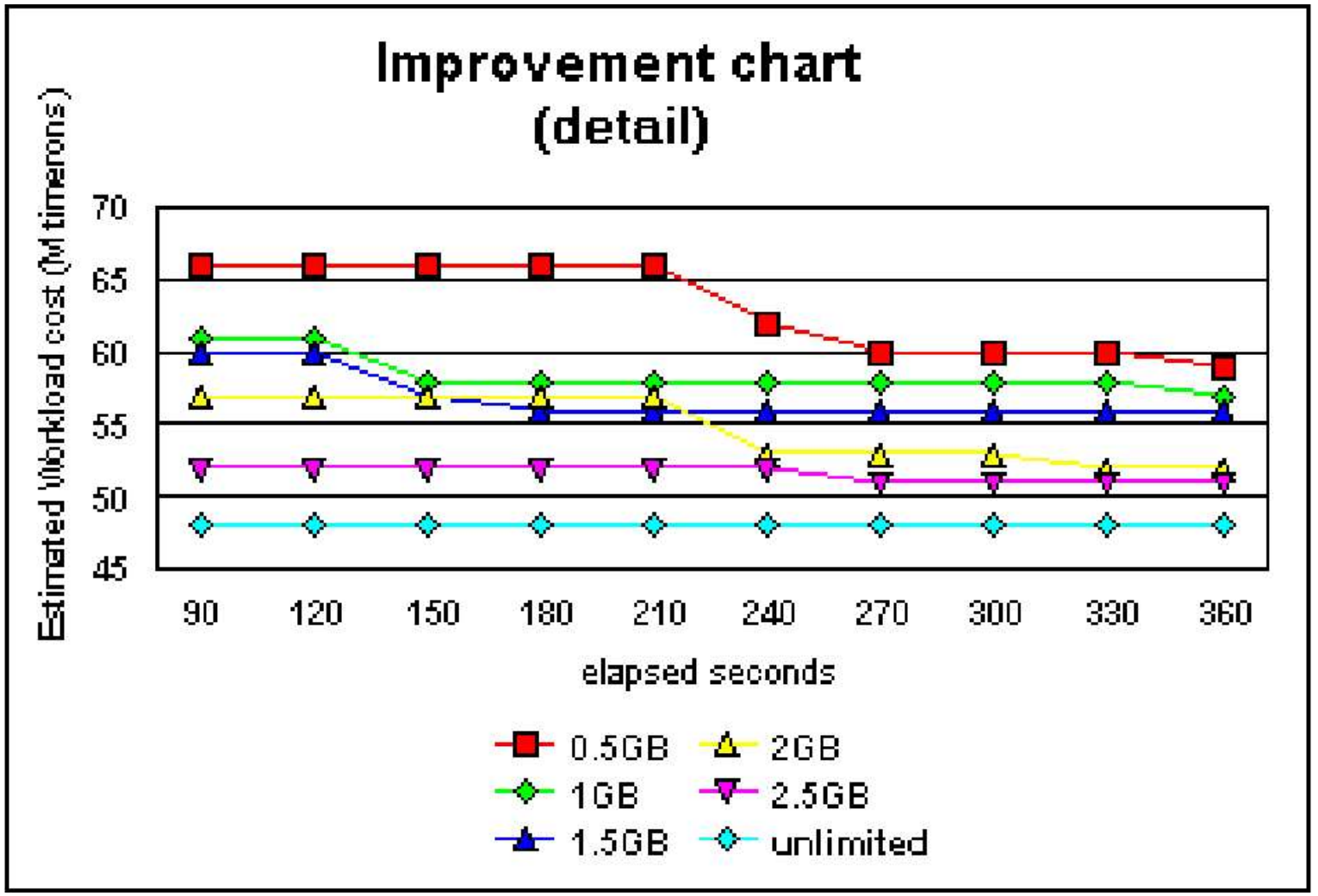
9. while (time did not expire) repeat

(a) TRY VARIATION

Test Results 1



Test Results 2



Conclusions

Microsoft SQL Server

- cost based
- works mainly outside DB optimizer
- reducing DB optimizer invocations
- extra tool

IBM DB2

- DB optimizer recommends indexes
- works mainly within the DB optimizer
- few optimizer call
- lower maintenance
- more complex DB optimizer

References

- [CN97] Surajit Chaudhuri and Vivek Narasayya, “An Efficient, Cost-Driven Index Selection Tool for Microsoft SQL Server”, Proceedings of the 23rd VLDB Conference Athens, Greece, 1997, pages 146-155
- [CN98] Surajit Chaudhuri and Vivek R. Narasayya, “AutoAdmin 'What-if' Index Analysis Utility”. In Haas and Tiwary [HT98], pages 367-378
- [LVZZS00] G. Lohman, G. Valentin, D. Zilio, M. Zuliani, A Skelly, "DB2 Advisor: An optimizer smart enough to recommend its own indexes", Proceedings, 16th IEEE Conference on Data Engineering, San Diego, CA, 2000.