Journal of Zhejiang University SCIENCE A
ISSN 1009-3095 (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Automatic relational database compression scheme design based on swarm evolution*

HU Tian-lei†, CHEN Gang, LI Xiao-yan, DONG Jin-xiang
(*School of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China*)
†E-mail: andy_hu@263.net

Received Dec. 11, 2005; revision accepted Apr. 5, 2006

Abstract: Compression is an intuitive way to boost the performance of a database system. However, compared with other physical database design techniques, compression consumes large amount of CPU power. There is a trade-off between the reduction of disk access and the overhead of CPU processing. Automatic design and adaptive administration of database systems are widely demanded, and the automatic selection of compression schema to compromise the trade-off is very important. In this paper, we present a model with novel techniques to integrate a rapidly convergent agent-based evolution framework, i.e. the SWAF (SWarm Algorithm Framework), into adaptive attribute compression for relational database. The model evolutionally consults statistics of CPU load and IO bandwidth to select compression schemas considering both aspects of the trade-off. We have implemented a prototype model on Oscar RDBMS with experiments highlighting the correctness and efficiency of our techniques.

Key words: Database compression, Automatic physical database design, Swarm evolution
doi:10.1631/jzus.2006.A1642 **Document code:** A **CLC number:** TP273; TP311.13

INTRODUCTION

As the Internet develops, the volume of information increases sharply. Capacity of storage medium increases exponentially, while the power of CPU chips increases under Moore's Law. However, transfer bandwidth of storage medium increases much slower. For a large relational database system (with capacity between a few giga- to multi-tera-bytes), bandwidth of the magnetic disks is always the bottleneck of performance. In traditional database research, researchers studied various physical database design methods, including indexes, materialized views, vertical/horizontal partitioning (Steve, 1993; Jeffrey *et al*., 2002) and compression (Severance, 1983) to solve this problem. Database compression boosts system performance by saving storage space and reducing disk access frequency. Many compression techniques for databases have been studied (Roth and van Horn, 1993; Goldstein *et al*., 1998), and some commercial DBMS introduced compression functions (Bruni and Naidoo, 1998; Oracle, 2002).

As features of relational database management system become more and more diversified and complicated, its management and optimization become much more difficult. Many applications require that a database system is simple to manage and that its performance is predictable. Furthermore, a self-managing/autonomic DBMS is very desirable. Nowadays, simplifying database management and automatically designing physical database schemas has become a hotspot in database research (Chaudhuri and Weikum, 2000; Agrawal *et al*., 2000; 2003; 2004a; 2004b; Zhang *et al*., 2001; Yu *et al*., 2003; Zilio *et al*., 2004). Database compression, distinct from techniques in those researches, demands additional CPU overhead, which means it will sharply increase CPU load while decreasing the data scale during the compressing process. The trade-off must

* Project (No. 2004AA4Z3010) supported by the National Hi-Tech Research and Development Program (863) of China

be taken into account to decide which compression schema provides the highest throughput.

The rest of this paper is organized as follows. We first review related work, and then formalize the "adaptive database attribute compression" problem. An evolution model based on SWAF (Xie and Zhang, 2004) is proposed to solve this problem, and we discuss two key algorithms in the model. We use a prototype of the model to compare results on different workloads. In the last section, we conclude the paper.

RELATED WORK

There are two advantages in compressing data in database systems: one is reduced database scale, and the other is improved performance. Due to inexpensive and popularization of mass storage devices, we only focus on the performance issue. Cormack (1985) presented a database compression method based on revised Huffman code. Graefe and Shapiro (1991) studied performance improvement when data are kept compressed as long as possible in query execution. Chen *et al*.(2001) and Hoque *et al*.(2002) continued the work by discussing the query optimization algorithms for compressed data. Roth and van Horn (1993) summarized various compression methods, and analyzed advantages and disadvantages of compression. Goldstein *et al*.(1998) divided database compression into two levels, page-level and file-level, and introduced a compression algorithm based on frames of reference. In the commercial world, Oracle 9i Release 2 introduced table compression features based on extracting same-value attributes in a data page (Oracle, 2002). IBM DB2 exploited hardware-based compression function in its OS/390 version (Bruni and Naidoo, 1998). The compression method discussed in this paper selects compression method for each attribute in each relation adaptively according to its type and value. The technique integrates the above compression algorithms' merits, and can be seamlessly integrated with the table compression feature provided by Oracle to offer better overall performance.

Due to the diversity of features and scale of data in a database system, it is increasingly imperative to automatically administer the database. Brown *et al*.(1994) considered automatic resource assignment algorithm to satisfy performance goals. Chaudhuri and Weikum (2000) analyzed the increasingly complicated architecture of a database system, and indicated that the architecture should have to be designed as a self-tunable RISC-style system. Agrawal *et al*.(2000; 2003; 2004b) studied automatic physical database design algorithms on index and materialized view selection, storage layout and partitioning. Commercial DBMSs, such as Oracle, Microsoft SQL Server and IBM DB2, introduced functions to design index and materialized view automatically (Agrawal *et al*., 2004a; Zilio *et al*., 2004). Zhang *et al*.(2001) and Yu *et al*.(2003) integrated evolution computation techniques into automatic materialized view selection. Chen (2002) designed a database system with automatic compression features; and concentrated on compressing string values using hierarchical dictionary-encoding methods, with the proposal providing very limited compression methods for automatic selection.

We identify the search for the optimal compression schema design as a discrete optimization problem. Evolution computation techniques can solve various discrete/continuous optimization problems, and are robust and converge rapidly. They succeed from three strongly related but independently developed approaches: Genetic Algorithm, Evolutionary Programming, and Evolutionary Strategies (Bäck, 1997). Researchers introduced the concept of swarm evolution (Steve, 1998) based on researches on biological social evolution. In swarm evolution, each individual in the society retrieves evolution knowledge in two ways: individual evolution and social evolution. Dorigo *et al* introduced ant colony optimization based on observation of the food-seeking strategy of a colony of ants (Dorigo *et al*., 1996; 1999; Dorigo and Gambardella, 1997). Kennedy studied the movement of a flock of birds and a school of fishes, and presented the particle swarm optimization (Kennedy and Eberhart, 1995; Clerc and Kennedy, 2002). Zhang and Xie (2003) introduced a swarm evolution algorithm based on differential evolution which exploits a differential vector studied by Storn and Price (1997). Based on previous research work, Xie and Zhang (2004) presented the SWAF evolution framework, which integrates different swarm evolution algorithms by different generate-and-test rules and adaptively deploys the rules. It was proved to be a robust and rapidly convergent evolution framework.

This paper integrates swarm evolution into adaptive database attribute compression. The optimization is triggered periodically or by a threshold value, and according to the statistics collected from DBMS and underlying OS, the evolution framework will run for a while. The optimal result, the best compression schema, will be generated and data in the database will be compressed directly or indirectly according to the scheme. Experiments on Oscar DBMS show that these techniques are valuable and heuristic for researches on database compression and automatic physical database design.

ADAPTIVE DATABASE ATTRIBUTE COMPRESSION (ADAC)

Various types of data are stored in database, and different compression algorithms generate different compression ratios. Even for the same type data, the compression ratio varies a lot. Moreover, compression generates high CPU overhead, and, for a system with high CPU load when nothing is compressed, compressing all data in the database is not a good idea. It is important to provide an adaptive way to manage the compression of database attributes. In database attribute compression, the database system adaptively generates compression schema according to current data schema, current workload and runtime environment, to provide the highest database throughput. We will first clarify some related concepts before digging into the details.

System Environment (SE) is the environment of a database system. $SE=<W, A, E>$, where W describes the workload, A is the database data schema and environment parameters E include CPU capability, I/O bandwidth, etc. Every work w ($w\in W$) is defined by a tuple $<q, f>$, where q is an SQL statement and f is the occurrence frequency of the statement. Each attribute a ($a\in A$) designates a column in a table or an index key.

Compression Method (CM) is a set of compression functions. For a given compression algorithm, if it generates different compress-ratios for the same data when configured with various arguments, it is treated as different compression functions. Compression Schema (CS) is a set of compression policies, with each compression policy $cs\in CS$ being defined by a tuple $<a, cm>$, where $a\in SE.A$, $cm\in CM$, and $\forall a\in SE.A$, $\exists cs\in CS$:$cs.a=a$. For generality, we ignore detailed compression functions when discussing the framework and optimization algorithms below. Instead, the involved compression algorithms will be mentioned when evaluating the proposed techniques.

Given certain SE, applying a CS, the throughput (TP) is defined as the number of processed SQL statements per time unit. $TP=tp_{SE}(CS)$, $TP\in\mathbb{R}$, and tp_{SE} is the function for computing throughput with SE. Finally, we define the ADAC problem as:

Definition 1 (Adaptive Database Attribute Compression Problem) Find a subscript vector $\boldsymbol{X}=(x_1, \ldots, x_i, \ldots, x_D)$ ($D=|SE.A|$, $1\le i\le D$, $x_i\in\mathbb{Z}$, $1\le x_i\le|CM|$) which constructs a compression schema $CS=\{(a_i, cm_{xi})|cm_{xi}\in CM, a_i\in SE.A, 1\le i\le|SE.A|\}$ to let $tp_{SE}(CS)$ be maximum.

AN ADAC MODEL BASED ON SWARM EVOLUTION

Swarm algorithm framework

A framework derived from SWAF (Xie and Zhang, 2004) is introduced to solve the ADAC problem.

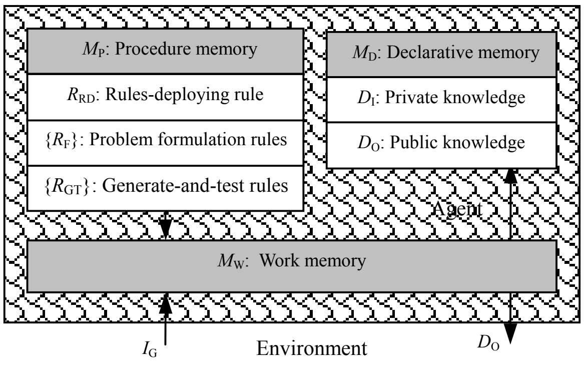
Definition 2 (Society) The society S is a multidimensional space for all agents to live and communicate. S is defined by a triple $<SS, Eval, I_G>$, where SS is the solution space of the problem, $Eval$ is an evaluation function to evaluate each position in SS and I_G is the shared information in the society. In ADAC problem, $SS=\{i|0\le i\le|CM|, i\in\mathbb{Z}\}^{|S.A|}$, $Eval=tp_{SE}$ and I_G is to store the global optimized solution.

Definition 3 (Agent) Each agent A_g is a cognitive unit in fast-and-frugal heuristics. Instead of communicating with each other directly, the agents communicate by reading or writing the global shared information $S.I_G$. Fig.1 is the architecture of the agent, and formally, A_g is defined by a triple $<M_P, M_D, M_W>$, where:

M_P is the Procedure Memory, which stores rules to control the evolution process. It is composed of three types of rules: $\{R_F\}$ is a set of rules to describe the constraint of a problem, $\{R_{GT}\}$ is a set of rules to generate and evaluate the target position of a movement and R_{RD} is a rule to deploy the above rules selectively.

M_D is the Declarative Memory, which stores knowledge possessed by an agent. It is divided into two categories: one is public knowledge D_O, and the other is private knowledge D_I. Only information in D_O will be updated to $S.I_G$. For different rules in M_P, the information in M_D is different; however, at least, the current position and the best position of this agent should exist in M_D.

M_W is the Work Memory. When an agent is activated, according to the deployed rule from M_P, it will compute in M_W with information from M_D and $S.I_G$. After new position is generated, the agent will move to the new position, and may update information to $S.I_G$.



M_P: Procedure memory
R_{RD}: Rules-deploying rule
$\{R_F\}$: Problem formulation rules
$\{R_{GT}\}$: Generate-and-test rules
M_D: Declarative memory
D_I: Private knowledge
D_O: Public knowledge
Agent
M_W: Work memory
I_G
Environment
D_O

Fig.1 Architecture of agents in the swarm algorithm framework

Definition 4 (SWAF) We define the swarm algorithm framework as a tuple $<S, Q>$, where S is the society and $Q=\{Ag_i|1\le i\le N, i\in\mathbb{Z}\}$ is composed of N agents.

Descriptions of the ADAC model

The ADAC model based on SWAF framework is shown in Fig.2. We study key components in ADAC model as follows:

Workload Miner (WLM) collects necessary knowledge of the environment. The knowledge comes from two sources. One is the database management system, including the workload (*SE.W*), data schema (*SE.A*), average CPU and I/O bandwidth usage for each attribute in *SE.A* and average compress-ratio and CPU overhead for each compression function in CM and each attribute in *SE.A*. The other one is the underlying operating system, including overall CPU and I/O bandwidth usage and memory usage. The collector periodically collects information from DBMS and OS, and forgets old information according to its "age".

Throughput Estimator (TPE) estimates throughput of a database system at given system environment *SE* and compression schema *CS*. TPE is initialized with information collected by EIC to generate tp_{SE},

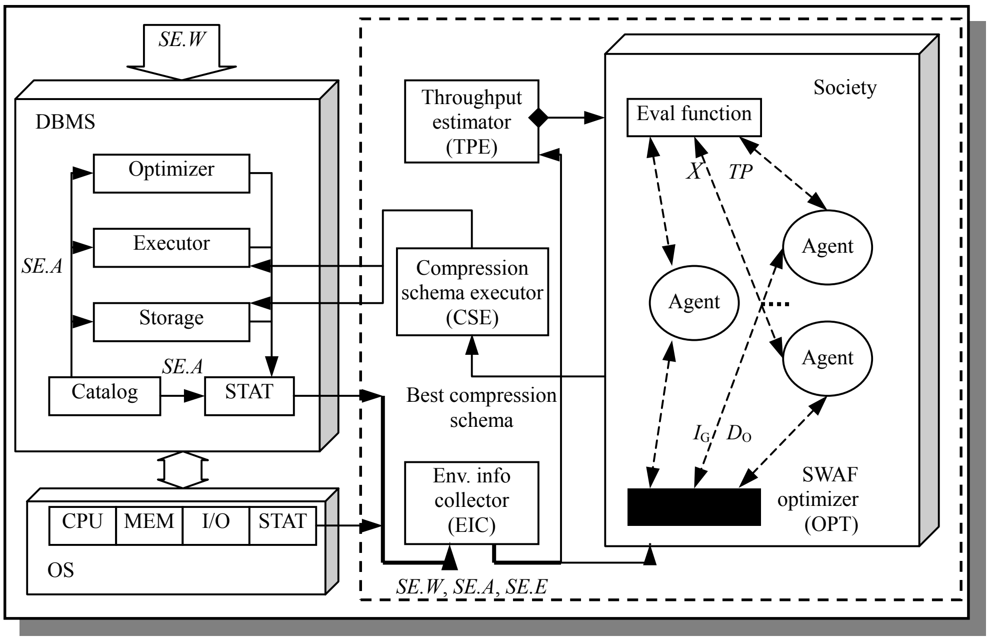


Fig.2 The ADAC model. Left part is the traditional architecture of a database system, and some irrespective database modules are omitted. Right part of the figure, in the dashed frame, is the key additional module appended to the database system by ADAC

and it will evaluate the evaluation request *X* from the SWAF optimizer and send the result $tp_{SE}(X)$ back to the optimizer.

Optimizer (OPT) is the key component in the model. Currently, it is implemented exploiting the SWAF framework. However, other evolution algorithms can simply replace it. OPT is initialized with knowledge collected by EIC, and activates a number of agents to start evolution. OPT contacts TPE constantly to evaluate positions in the solution space, and, after a few iterations, the best result, i.e. the compression schema with maximal throughput, is sent to compression schema executor (CSE).

CSE employs optimal compression schema in the database system. CSE works in two ways, one is directly compressing, and another is indirectly compressing. For the first, CSE directly accesses storage module in DBMS to compress data. It only affects physical structures of an attribute value, but not the logical semantics, so only page latches other than transaction level locks (Gray and Reuter, 1993) are required. For the second way, CSE delegates compression to executor module in DBMS. Whenever $SE.W$ alters an attribute, the executor will compress it before storing it according to the compression schema.

While running, EIC continuously collects environment information, and activates optimization conditionally (for example, a period of time elapses from last activation, or the system load reaches a threshold). When optimization starts, first, TPE generates evaluation function tp_{SE}, and OPT initializes its inner SWAF framework, setting up the society, the number of agents and deploying agents at different positions in the society. Consequently, OPT activates all agents to let them roam in the solution space according to the evolution algorithm. After a few cycles of optimization, an optimal solution is stored in $S.I_G$, and OPT stops optimization and submits the solution to CSE. CSE executes directly or indirectly to compress data accordingly.

Compared with traditional database compression and automatic database administration techniques, the model presented above has many advantages.

First, the model utilizes an adaptive framework, and will be activated under certain circumstances. The compression schema will be dynamically adjusted when the workload, the data schema or the other environment factors changes. The procedure will be automatically performed without administrators.

Second, the model will compress data in the background, and will not markedly affect the performance and simultaneousness of normal transaction processing.

Third, components in the model are conceptually independent and logically decoupled from the DBMS. Benefiting from the independence, these components may run on other machines, and communicate with the database system through network. The competition for resources between normal transaction processing and optimization can be alleviated.

Last, the optimization algorithm in the model is based on the SWAF swarm evolution framework. The agents live in the society move towards a direction chosen according to both the optimal solution of the agent (local optimal solution) and the optimal solution of the society (global optimal solution). Due to the R_{RD} rule deploying various generate-and-test rules in $\{R_{GT}\}$ adaptively, the model will take advantage of various swarm optimization algorithms to achieve faster convergence.

KEY ALGORITHMS

This section will describe two major algorithms in ADAC model: the evolution algorithm in OPT and the throughput evaluation algorithm in TPE.

The SWAF evolution algorithm, responsible for solving the ADAC problem, is the most important algorithm in the model. The SWAF framework may utilize various optimization rules, but for convenience, we only use the rule for particle swarm algorithm (Kennedy and Eberhart, 1995). In this rule, three kinds of information, the current position, the last position in D_I and the local optimal position in D_O, are stored in M_D of each agent. In addition, the global optimal position is stored in $S.I_G$.

Algorithm 1 Denote N as the number of agents in SWAF, T as the maximum iteration count. At the tth iteration ($1\le t\le T$, $t\in\mathbb{Z}$), agent Ag_i ($1\le i\le N$, $i\in\mathbb{Z}$) is at position $X^{(t)}$ and the last position is $O^{(t)}$. The local optimal position is $P^{(t)}$, and the global optimal position is $G^{(t)}$.

1. Begin. Initialize the SWAF framework, and

randomly choose the initial position $X^{(1)}$ of each agent Ag_i in $SWAF.Q$. Let $P^{(1)}=O^{(1)}=X^{(1)}$, and $G^{(1)}=\max(X^{(1)}, Ag_i)$. Set iteration count t=1.

2. While $t\leq T$, set current agent number i=1, and begin an iteration.

3. While $i\leq N$, process the ith agent Ag_i.

4. Retrieve and activate the $R_{\rm PS}$ representing the particle swarm rule from $Ag_i.M_{\rm P}.R_{\rm GT}$, and retrieve $X^{(t)}$, $O^{(t)}$, $P^{(t)}$ from $Ag_i.M_{\rm D}$. Get $G^{(t)}$ from $S.I_{\rm G}$, and put all the information into $M_{\rm W}$.

5. Generate the next position of $X^{(t)}$, and calculate its dth dimension as in (Clerc and Kennedy, 2002):

$$X_d^{(t+1)}=|X_d^{(t)}+CF\cdot[V_d^{(t)}+(c_1+c_2)\cdot U_{\rm R}\cdot(P_d^{(t)}-X_d^{(t)})]|, \quad (1)$$

where $CF=2/\{[\varphi\cdot(\varphi-4)]^{1/2}+\varphi-2\}$, $\varphi=c_1+c_2>4$, $V^{(t)}=X^{(t)}-O^{(t)}$, and U_R is a random real value between 0 and 1. The default values for utilities are $c_1=c_2=2.05$.

6. Set $O^{(t+1)}=X^{(t)}$ to save the original position.

7. $P^{(t+1)}=\max(P^{(t)}, X^{(t+1)})$, Store $X^{(t+1)}$, $O^{(t+1)}$, $P^{(t+1)}$ back into $Ag_i.M_{\rm D}$.

8. $G^{(t+1)}=\max(G^{(t)}, X^{(t+1)})$, set $G^{(t+1)}$ back to $S.I_{\rm G}$.

9. i++. End While (for i)

10. t++. End While (for t)

11. Return $G^{(t)}$ in $S.I_{\rm G}$, End.

The throughput estimate algorithm is used to evaluate solutions during the evolution procedure. It is difficult to estimate the throughput of a database system directly. The optimizers of traditional DBMS can only assess the cost for a single statement, but cannot estimate the impact of concurrent transactions, randomly or sequentially IOs, etc. The adopted estimate algorithm in ADAC model is according to historical information of a database system. Although there are limitations of this approach, results of experiments proved it is accurate enough for ADAC model in most applications.

Algorithm 2 For an attribute a_i in $SE.A$, using function cm_j in CM to compress it, the CPU overhead ratio is $f_{\rm CPU}(i,j)\geq1$, and the IO reduction ratio is $f_{\rm IO}(i,j)\geq1$. Compute the throughput for a given solution X as follows:

1. Begin. Initialize TPE. According to the statistics, for a unit of TP, compute the percentage of CPU load and I/O amount needed to process each attribute $a_{\rm i}$: $cpu_{\rm i}$ and $io_{\rm i}$. Also, compute the percentages of CPU load and IO caused by other processes: $cpu_{\rm o}$ and $io_{\rm o}$.

2. Applying compression schema CS, the CPU load needed to process $a_{\rm i}$ is $cpu_{\rm i}\cdot f_{\rm CPU}(i, X_{\rm i})$, and the needed IO amount is $io_{\rm i}\cdot f_{\rm IO}(i, X_{\rm i})$.

3. Estimate percentages of CPU and IO for a unit of TP as follows:

$$\left.\begin{aligned} CPU_x &= \sum_{i=1}^{N}[cpu_{\rm i}\cdot f_{\rm CPU}(i,X_{\rm i})]+cpu_{\rm o},\\ IO_x &= \sum_{i=1}^{N}[io_{\rm i}\cdot f_{\rm IO}(i,X_{\rm i})]+io_{\rm o}. \end{aligned}\right\} \quad (2)$$

4. Respectively evaluate TP when CPU or IO is the bottleneck: $TP_{\rm CPU}=\alpha/CPU_x$ and $TP_{\rm IO}=\beta/IO_x$, where α and β are the bottleneck percentage parameters for CPU and IO. According to the experiment results, we let $\alpha=\beta=80\%$ by default.

5. Return $\min(TP_{\rm CPU}, TP_{\rm IO})$, End.

For estimating the CPU load and I/O amount per attribute in Algorithm 2, we implemented additional statistics on average length of an attribute in a relation, average length of a row in a relation, and average processing count of each attribute in each relation in a workload.

EXPERIMENTS

Setup

The authors developed a prototype system of ADAC model on Oscar DBMS. Following experiments were all based on Oscar DBMS V5.5 for Redhat Linux 9.0. CPU hyper-threading was turned off, and write cache of disk device was turned off either. All experiments were done by running the system for a period, activating the optimization process for 1000 iterations, and collecting statistics after compression finished.

Following tables show the experiment configurations. Table 1 is the hardware configurations, while Table 2 is the experiment workloads.

We provide compression algorithms for most common types in the DBMS. For instance, attributes with BPCHAR/VARCHAR types are associated with Huffman encoding, LZ, bit cutting (i.e., for an all ASCII value, we transform each character from 8 bits to 7 bits; and for an all digit value, we apply an 8 to 4

Table 1 Hardware configurations used in the experiments

PLM No.	Machine	CPU	Memory	Storage device
A	Dawning S240XP Server	Xeon 2 G CPU*4 SMP	2 GB	Dual SCSI Disk RAID-0
B	PC Work-station	Pentium IV 2.4 G CPU*1	512 MB	IDE Disk

Table 2 Experiment workloads

Group	PLM No.	Experiment workload
1	A	TPC-C 120W
2	B	TPC-C 15W
3	A	TPC-W 2000 EB
4	B	TPC-W 200 EB
5*	A	PDM Brute Force

TPC-C (TPC, 2005) and TPC-W (TPC, 2002) are international benchmarks for database systems. *: based on a PDM application system developed by us, and the corresponding workload is a group of SQL statements simulating the daily operations in a brute-force way

bits compression), space trimming (removing all padding blanks from a BPCHAR attribute). For an INT2/INT4/INT8 attribute, we compress the values to fewer bytes than it should have (1 byte for $0\sim2^7-1$, 2 bytes for $2^7\sim2^{15}-1$, etc.). We sort the compression methods by their data type and their compression ratio in compression functions set *CM*.

Evaluate the ADAC model

Figs.3~6 show comparisons of data scales, CPU loads, IO amount and throughput in three situations. The first is not to compress at all (NONE), the second is to compress with ADAC model (ADAC) and the last is maximally compressing all data in the database (MAXIMIZE). For conciseness, we omit indexes created for each workload.

As shown in Fig.3, for every workload, the compress-ratio: MAXIMIZE>ADAC>NONE. Fig.4 shows that when nothing is compressed, the experiments for 3 and 4 have relatively high CPU loads. If these workloads are compressed excessively, CPU may become the bottleneck of performance, and the overall performance drops. When applying ADAC, only a few attributes which have relatively considerable impact on IO will be compressed. The CPU loads after compression are almost the same as the original as shown in Fig.4.

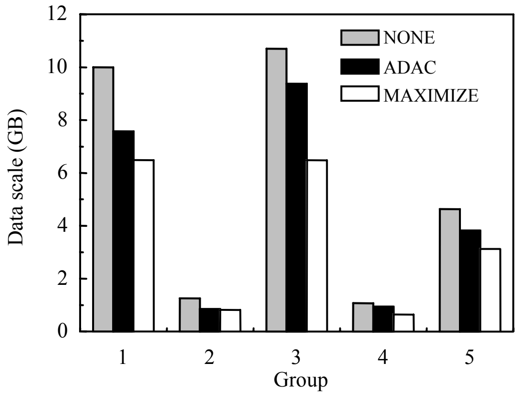


Fig.3 Comparison of data scales

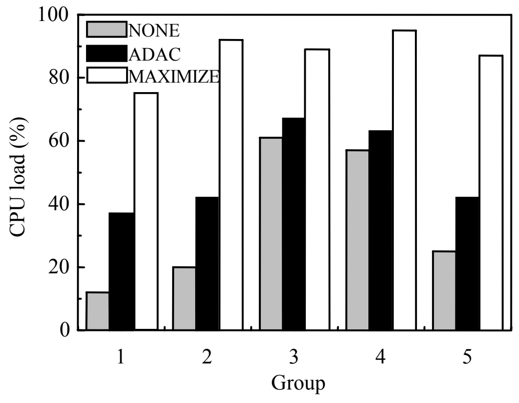


Fig.4 Comparison of CPU loads

In Fig.5, after ADAC compression, the I/O amount is less than NONE due to reduction in data scales. However, when applying MAXIMIZE compression, the bottlenecks are no longer I/O, and the amount of I/O decreased distinctly. In the fifth workload, there are large numbers of sequential I/O's, obviously larger than other workloads. Because of the incomparability of the workloads, the throughputs in Fig.6 are normalized before comparison, e.g., the throughput of each workload is fixed at 1.0 for NONE, and the computed ratio of ADAC and MAXIMIZE's throughput are compared to it. The results in Fig.6 show that ADAC-compression provides better throughput compared with NONE and MAXIMIZE.

Parameter selection

Tunable parameters in ADAC model are α and β in the throughput estimator algorithm.

In Fig.7, fix β=80% and vary α in [40%, 120%] in steps of 10%. When α nears 80%, the throughput reaches the maximum. To let $\alpha\leq$60% leads to under-

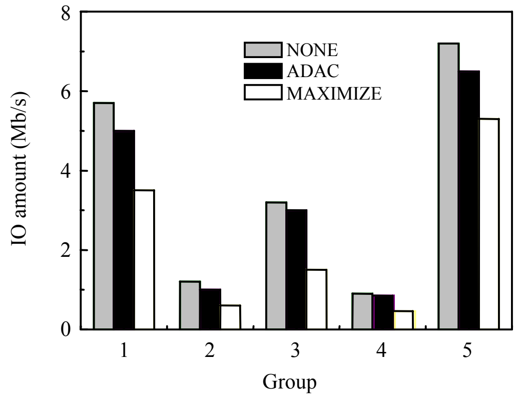


Fig.5 Comparison of I/O amount

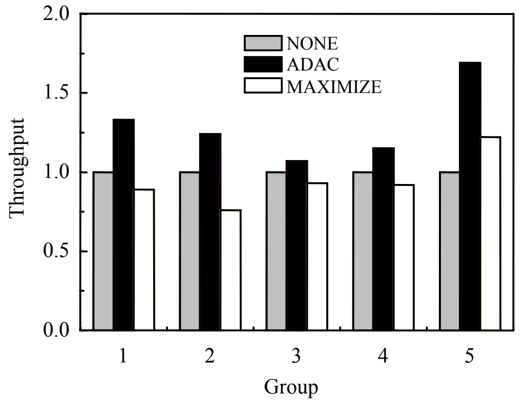


Fig.6 Comparison of normalized throughput

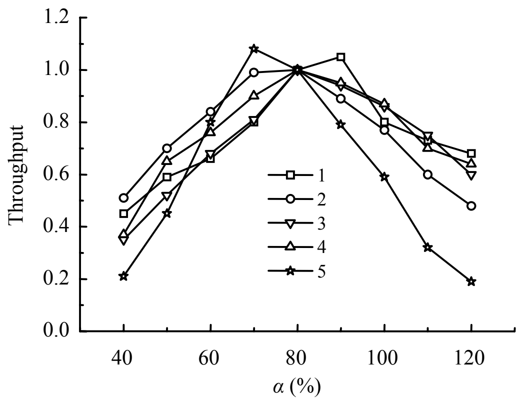


Fig.7 Impacts on throughput by different *α*'s

estimating the CPU power, the compression schema is not optimal, and the performance drops. To let $\alpha \geq 100\%$ leads to overestimating the CPU power, the compression schema excessively compressed data, and CPU becomes the bottleneck to decrease the performance sharply.

In Fig.8, fix α=80% and vary β in [40%, 120%] in steps of 10%. When β is larger than 70%, the throughput tends to be an invariable maximal value. To let $\alpha \leq 60\%$ leads to underestimating the IO bandwidth and to compressing the database excessively. However when $\alpha \geq 80\%$, properly or over estimating the IO bandwidth, the performance estimation is only related with the CPU power and not related with β. So, the generated compression schema and the optimal throughputs are all the same.

Experiments revealed that when $\alpha=\beta$=80%, the ADAC model will mostly boost the overall performance of a database system.

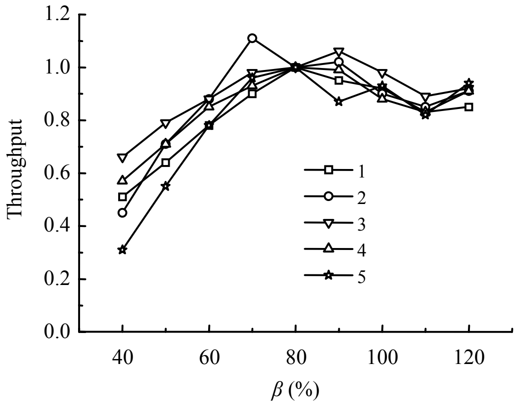


Fig.8 Impacts on throughput by different *β*'s

CONCLUSION

In this paper, we introduce the adaptive database attribute compression (ADAC) problem. A model which adopts a swarm evolution algorithm based on the SWAF framework is proposed to solve the problem. The model is prototyped, with experiments showing that ADAC is correct and efficient for automating the selection of compression scheme. The contribution of our work includes:

(1) Introducing the adaptive database attribute compression;

(2) Proposing a physical database design framework aiming at solving not only the database compression problem, but also other physical database design problems;

(3) Integrating swarm evolution to do optimal search in a huge solution space introduced by the adaptive database attribute compression problem.

Future researches include a theoretical model to estimate the performance by mining statistics and trying to apply our techniques to solve other automatic physical database design problems.

References

Agrawal, S., Chaudhuri, S., Narasayya, V., 2000. Automated Selection of Materialized Views and Indexes for SQL Databases. 26th International Conference on Very Large Databases. Morgan Kaufmann, Cairo, p.496-505.

Agrawal, S., Chaudhuri, S., Das, A., Narasayya, V., 2003. Automating Layout of Relational Databases. 19th International Conference on Data Engineering. IEEE Computer Society, Bangalore, p.607-618.

Agrawal, S., Chaudhuri, S., Kollar, L., Marathe, A., Narasayya, V., Syamala, M., 2004a. Database Tuning Advisor for Microsoft SQL Server 2005. 30th International Conference on Very Large Databases. Morgan Kaufmann, Toronto, p.1110-1121.

Agrawal, S., Narasayya, V., Yang, B., 2004b. Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design. ACM SIGMOD International Conference on Management of Data. ACM, Paris, p.359-370. [doi:10.1145/1007568.1007609]

Bäck, T., 1997. Evolutionary computation: comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, **1**(1):3-17. [doi:10.1109/4235.585888]

Brown, K.P., Mehta, M., Carey, N.J., Livny, M., 1994. Towards Automated Performance Tuning for Complex Workloads. 20th International Conference on Very Large Databases. Morgan Kaufmann, Santiago de Chile, p.72-84.

Bruni, P., Naidoo, R., 1998. DB2 for OS/390 and Data Compression. IBM Redbook.

Chaudhuri, S., Weikum, G., 2000. Rethinking Database System Architecture: Towards a Self-tuning RISC-style Database System. 26th International Conference on Very Large Databases. Morgan Kaufmann, Cairo, p.1-10.

Chen, Z.Y., 2002. Building Compressed Database Systems. Ph.D Dissertation, Cornell University.

Chen, Z.Y., Gehrke, J., Korn, F., 2001. Query Optimization in Compressed Database Systems. ACM SIGMOD International Conference on Management of Data. ACM, Santa Barbara, p.271-282.

Clerc, M., Kennedy, J., 2002. The particle swarm—explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, **6**(1):58-73. [doi:10.1109/4235.985692]

Cormack, G.V., 1985. Data compression on a database system. *Communications of ACM*, **28**(12):1336-1342. [doi:10.1145/214956.214963]

Dorigo, M., Gambardella, L.M., 1997. Ant colonies for the traveling salesman problem. *Biosystems*, **43**(2):73-81. [doi:10.1016/S0303-2647(97)01708-5]

Dorigo, M., Maniezzo, V., Colorni, A., 1996. The ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Csybernetics, Part B*, **26**(1):29-41. [doi:10.1109/3477.484436]

Dorigo, M., Gianni, D.C., Gambardella, L.M., 1999. Ant algorithms for discrete optimization. *Artificial Life*, **5**(2):137-172. [doi:10.1162/106454699568728]

Goldstein, J., Ramakrishnan, R., Shaft, U., 1998. Compressing Relations and Indexes. 14th International Conference on Data Engineering. IEEE Computer Society, Orlando, p.370-379. [doi:10.1109/ICDE.1998.655800]

Graefe, G., Shapiro, L.D., 1991. Data Compression and Database Performance. Symposium on Applied Computing. IEEE Computer Society, Kansas, p.22-27.

Gray, J., Reuter, A., 1993. Transaction Processing: Concepts and Techniques. Morgan Kaufmann.

Hoque, A.S.M.L., McGregor, D.R., Wilson, J., 2002. Database Compression Using an Offline Dictionary Method. 2nd International Conference on Advance in Information Systems. Springer-Verlag Heidelberg, Izmir, p.11-20.

Jeffrey, A.H., Mary, P., Fred, R.M., 2002. Modern Database Management (6th Ed.). Prentice Hall.

Kennedy, J., Eberhart, R., 1995. Particle Swarm Optimization. International Conference on Neural Network. Institute of Electrical and Electronics Engineers, Perth, p.1942-1948.

Oracle, 2002. Table Compression in Oracle9i Release2. Oracle Technical White Paper. Available at http://www.oracle.com/technology/products/bi/pdf/o9ir2_compression_twp.pdf on 21st Feb., 2006.

Roth, M.A., van Horn, M.J., 1993. Database compression. *ACM SIGMOD Record*, **22**(3):31-39. [doi:10.1145/163090.163096]

Severance, D., 1983. A practitioner's guide to data base compression. *Information Systems*, **8**(1):51-62. [doi:10.1016/0306-4379(83)90030-3]

Steve, R., 1993. Automating Physical Database Design. Ph.D Dissertation, Department of Computer Science of the Graduate School of Arts and Sciences, New York University.

Steve, A.F., 1998. Foundations of Social Evolution. Princeton University Press.

Storn, R., Price, K., 1997. Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, **11**(4):341-359. [doi:10.1023/A:1008202821328]

TPC, 2002. TPC Benchmark W (Web Commerce) Specification (Version 1.8). Transaction Processing Performance Council. Available at http://tpc.org/tpcw/spec/tpcw_V1.8.pdf on 21st Feb., 2006.

TPC, 2005. TPC Benchmark C Standard Specification (Revision 5.6). Transaction Processing Performance Council. Available at http://tpc.org/tpcc/spec/tpcc_current.pdf on 21st Feb., 2006.

Xie, X.F., Zhang, W.J., 2004. SWAF: Swarm Algorithm Framework for Numerical Optimization. *In*: Deb, K., Poli, R., Banzhaf, W., *et al*. (Eds.), Genetic and Evolutionary Computation Conference. Springer-Verlag Heidelberg, Seattle, p.238-250.

Yu, X.J., Yao, X., Choi, C.H., Gou, G., 2003. Materialized view selection as constrained evolutionary optimization. *IEEE Transaction on System, Man and Cybernetics, Part C*, **33**(4):458-467. [doi:10.1109/TSMCC.2003.818494]

Zhang, W.J., Xie, X.F., 2003. DEPSO: Hybrid Particle Swarm with Differential Evolution Operator. IEEE International Conference on Systems, Man and Cybernetics. IEEE Computer Society, Washington D.C., p.3816-3821.

Zhang, C., Yao, X., Yang, J., 2001. An evolutionary approach to materialized view selection in a data warehouse environment. *IEEE Transaction on System, Man and Cybernetics, Part C*, **31**(3):282-294. [doi:10.1109/5326.971656]

Zilio, D.C., Zuzarte, C., Lightstone, S., *et al*., 2004. Recommending Materialized Views and Indexes with IBM DB2 Design Advisor. International Conference on Autonomic Computing. New York, p.180-188.