Online monitoring and visualisation of database structural deterioration

Takashi Hoshino*, Kazuo Goda and Masaru Kitsuregawa

Institute of Industrial Science,
The University of Tokyo,
4-6-1 Komaba, Meguro-ku, Tokyo, 153-8505, Japan
E-mail: hoshino@tkl.iis.u-tokyo.ac.jp
E-mail: kgoda@tkl.iis.u-tokyo.ac.jp
E-mail: kitsure@tkl.iis.u-tokyo.ac.jp
*Corresponding author

Abstract: Structural deterioration of database is caused by updates of the database and can degrade performance seriously. Database reorganisation, which removes structural deterioration by relocating data in the secondary storage, is an administrator's headache; it is difficult to schedule reorganisation efficiently to keep acceptable performance because conventional database systems lack crucial monitoring facility of structural deterioration, with high resolution, high accuracy and low overhead. In order to make database reorganisation be easy for administrators and be autonomic in the future, the paper proposes an online database structure deterioration monitor, which can visualise structural deterioration distribution, with a novel structural deterioration model, a storage performance model and an incremental update technique. The experimental results with our prototype show that our scheme can monitor structural deterioration with an error of within 12% and with an overhead of 3% at most.

Keywords: visualisation; structural deterioration; database reorganisation; IO cost estimation; storage performance model; database system; online monitoring.

Reference to this paper should be made as follows: Hoshino, T., Goda, K. and Kitsuregawa, M. (2010) 'Online monitoring and visualisation of database structural deterioration', *Int. J. Autonomic Computing*, Vol. 1, No. 3, pp.297–323.

Biographical notes: Takashi Hoshino received his ME in Information and Communication Engineering from the Graduate School of Information Science and Technology of the University of Tokyo. He is currently a member of Generic Solution Corporation (hoshino.takashi@u-gs21.com). He was a JSPS Research Fellow (DC) in 2006 and 2007. His research interests are database and storage systems. He is a member of Information Processing Society of Japan and the Database Society of Japan.

Kazuo Goda received his BE, ME and PhD from the University of Tokyo in 2000, 2002 and 2005 respectively. He is currently a Project Research Associate at the Institute of Industrial Science, the University of Tokyo. His research interests include high performance database systems and highly functional storage systems. He is a member of ACM, IEEE Computer Society, USENIX, Information Processing Society of Japan and the Database Society of Japan.

Masaru Kitsuregawa received his PhD from the University of Tokyo in 1983. He is currently a Professor and the Director of the Center for Information Fusion at the Institute of Industrial Science of the University of Tokyo. His current research interests include database engineering, web mining and high performance database engine. He had been a VLDB Trustee and served as the Co-General Chair of IEEE ICDE 2005 at Tokyo. He is currently a Secretary of the IEEE Technical Committee on Data Engineering. He serves as a Science Advisor of Ministry of Education. He obtained the ACM SIGMOD E.F.Codd Innovation Award in 2009.

1 Introduction

The data storage structure of database is likely to deteriorate as records are inserted, updated or deleted many times. Such a phenomenon is called structural deterioration, which usually degrades performance of transaction and query processing. Database reorganisation, a unique solution to structural deterioration, moves records in the storage space to remove structural deterioration. Reorganisation is recognised as an essential function for database systems. In fact, all of the major database systems have incorporated reorganisation tools in their software suites.

Database administrators are responsible of carefully scheduling a reorganisation task in order to manage the system performance; specifically, the administrator should determine the portion of the database to be reorganised and the time point to trigger reorganisation. Database reorganisation is generally time consuming, issuing a massive number of disk accesses, thus it is likely to impact on online workloads. The administrator must consider current deterioration degree and also possible performance degradation caused by database reorganisation.

However, so far such elegant management of structural deterioration does not look feasible in practices. In general, database systems usually abstract data storage structures. This is beneficial to the users; they do not have to consider physical data storage and they can process and manage data by the use of higher-level representation such as SQL. At the same time, the database storage engine is also a black box to the administrator. Structural deterioration is also invisible in many cases although the information of structural deterioration is crucial to optimise database reorganisation. To date, many database administrators do not care about structural deterioration and then experience unexpected performance degradation that are really caused by structural deterioration. Even educated administrators of the high-end system may tend to make reorganisation plans based on rules of thumb by using rough performance statistics of the database. This kind of naive solutions leads to inefficient and expensive database administration. To solve this issue is a big challenge for autonomic database management.

The paper proposes an online monitoring facility of database structural deterioration, which we developed for open-source database systems. The monitor has the capability of keeping track of structural deterioration at any time. The monitor is comprised of three components, sniffer, estimator, and visualiser. The sniffer, which can be plugged into the storage engine of database system, is able to trace update events of the database physical structure and inform the estimator of them. The estimator, which is a dedicated management process, is able to calculate structural deterioration degree of the database on the basis of the noticed events. The visualiser, which is a GUI tool, is able to visualise the measured structural deterioration in the real-time fashion. All the components can work together so that structural deterioration can be observed precisely with little interference with transaction and query processing. Our contribution can make it easier for the administrator to understand the deteriorated portion of the database and the degree of performance impact. Much faster diagnosis of structural deterioration and more efficient reorganisation scheduling can be greatly supported. To the best of our knowledge, similar works have not been published or presented.

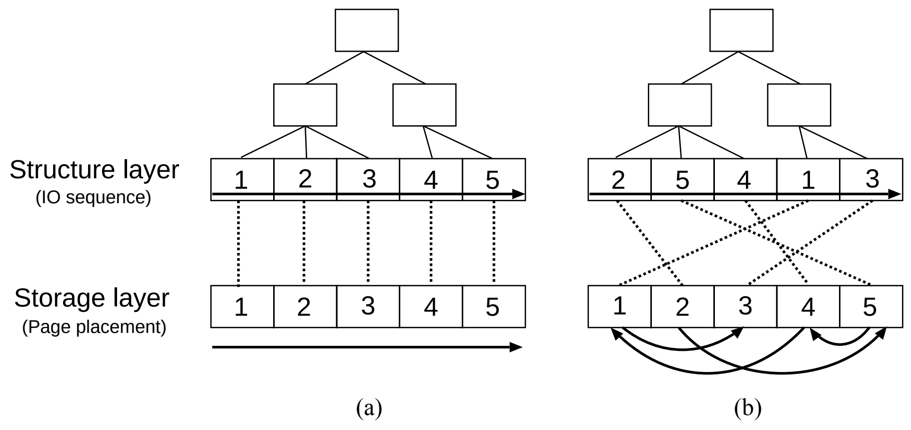
The rest of the paper is organised as follows. Section 2 describes structural deterioration more carefully and our autonomic management framework of structural deterioration. Section 3 presents a design of our online monitoring facility and its monitoring techniques. Section 4 shows our prototype implementation for open-source database systems and Section 5 evaluates our proposal. Related works are briefly presented in Section 6 and the paper is concluded in Section 7.

2 Structural deterioration

2.1 Definition of structural deterioration

Database structural deterioration is a phenomenon where data storage structure of the database deteriorates due to many records updates. Let us suppose B+tree (Bayer and McCreight, 1972; Bayer and Schkolniek, 1977; Comer, 1979; Lehman and Yao, 1981), a typical data structure that is used in many database systems. B+tree is mainly used for clustering records based on their key values. That is, in the most preferable cases, records are placed in key order in leaf pages as illustrated in Figure 1(a), so that those records can be scanned quickly in key order. This is beneficial, for example, for range scan operations that are often seen in many adhoc queries. However, the database is usually updatable. Records can be inserted, updated and deleted. Due to a number of such record manipulations, the stored records get gradually disordered. In a deteriorated case, for the same key-ordered record scan, pages are accessed in back and forth manner as shown in Figure 1(b), resulting in poor access performance.

Figure 1 Example of structural deterioration, (a) best-organised structure (b) deteriorated structure



In this paper, we formalise the degree of structural deterioration based on the performance degradation caused by the structural deterioration. Please assume a data set *D* is stored in the database and a query set *Q* is issued for *D* on the database. The degree of structural deterioration can be calculated in the following formula.

$$SD = \frac{C(D,Q)}{C_{min}(D,Q)}$$

Here, $C(D, Q)$ denotes query processing cost of *Q* if *Q* is executed for *D* in the current database, whereas $C_{min}(D, Q)$ denotes possible minimum cost if *Q* were executed for *D* in the best-organised database. That is, *SD* means the increase ratio of query processing cost caused by structural deterioration.

2.2 Difficulty of structural deterioration management

Database systems are actually used in many places for supporting our digitised business and social life. Since performance is a prime concern on top-end IT systems, structural deterioration is widely recognised as one of the most serious issues for database administration. The administrator is responsible of managing such deteriorating phenomena appropriately by the use of database reorganisation (Sockut and Goldberg, 1979). Specifically speaking, the administrator has to reorganise the database in order to recover originally expected performance only when the performance has degraded or is likely to degrade soon. However, the method of directly observing database structural deterioration was not established so far. It is very difficult for the administrator to identify structural deterioration when he/she observes performance degradation. Worse, database reorganisation is much data intensive, issuing a number of disk accesses to move many pages in the database. While the database is being reorganised, query and transaction processing may stall or its performance may degrade. The database administrator must also consider such temporal negative impact for scheduling reorganisation (Sockut et al., 1997).

The problem looks too hard, so many administrators do not consider such structural deterioration much carefully; database reorganisation is not triggered at all, or can be kicked on manually by the administrator based on rule of thumbs or rough performance statistics. Along such naïve solutions, the administrators may experience unexpected serious performance degradation caused by structural deterioration, and may degrade service availability due to unnecessary reorganisation.

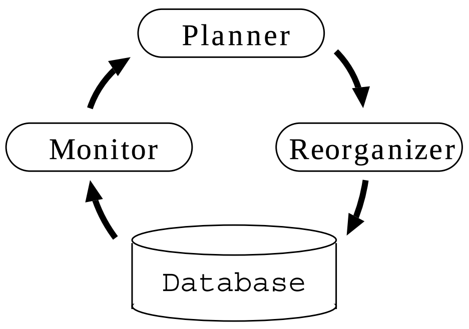
2.3 A framework for autonomic structural deterioration management

If structural deterioration could be managed in autonomic fashion, the benefit would be significant. That is, structural deterioration could be precisely measured at runtime, possible performance degradation could be estimated. Reorganisation could be more efficiently triggered. Reorganisation could be executed appropriately when structural deterioration is likely to degrade the database performance. In addition, deteriorated portion of the database could be identified, so that partial solution of reorganising only the deteriorated portion could decrease reorganisation cost.

Studying the feasibility and the potential benefit of such an autonomic solution is a big research challenge. Autonomic computing technologies will be strongly required by almost IT systems in the world over the future, considering that Kephart and Chess (2003) have said. This thought is especially true for database systems and storage systems, accordingly, various approaches are under investigation. Lightstone et al. (2002, 2003) has described their plans and activities toward autonomic database systems with their commercial product of database management software. The details have been discussed by Zilio et al. (2003, 2004), Telford et al. (2003) and Markl et al. (2003). In terms of storage systems, Ganger et al. (2003) and Uttamchandani et al. (2004) etc. are working on autonomic storage.

Figure 2 shows a framework of enabling autonomic management of structural deterioration. The framework is comprised of three key components, *monitor*, *planner* and *reorganiser*. The monitor keeps track of structural deterioration of the database online, and informs the planner of monitored information. The planner schedules reorganisation task; the target database section to be reorganised and the point in time to trigger reorganisation is determined based on the monitored information. The reorganiser reorganises the database based on the scheduled plan. Autonomic management of database structural deterioration is achieved by running this cycle continuously, not depending on any human administrator. The administrator might not have to consider reorganisation at all, or his/her administration burden of reorganisation could be significantly relived.

Figure 2 A framework for autonomic structural deterioration management



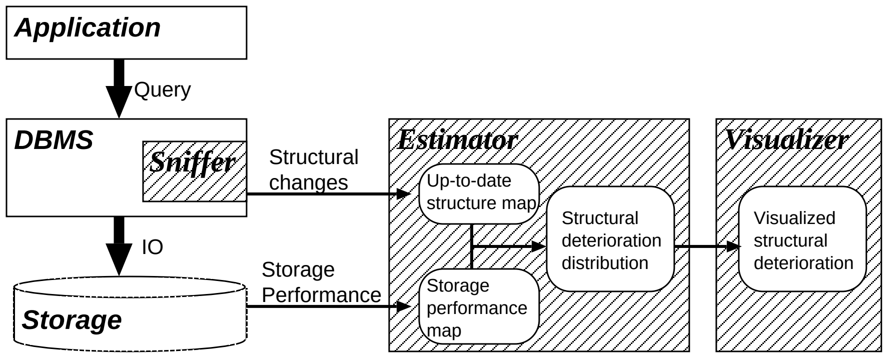
In this paper, we discuss a monitoring facility of structural deterioration as a first step toward autonomic management of structural deterioration. To the best our knowledge, no works attempted to directly monitor structural deterioration online. Our contribution opens a gate for new management paradigm of database structural deterioration.

3 Proposal

3.1 Design overview

In this section, we show an overview of proposed online database structural deterioration monitor, called *SD-Mon*. The design architecture is illustrated in Figure 3.

Figure 3 Design architecture of SD-Mon



The monitor consists of a *sniffer*, an *estimator*, and *visualiser*. The sniffer, implemented in low level storage engine of database systems, captures events that modifies data storage structures in the database selectively and sends the events to the estimator. The estimator keeps track of the up-to-date database structure based on the received information and estimates the degree of structural deterioration using our proposed structural deterioration metrics, and our proposed storage performance model. The visualiser makes panoramic views of structural deterioration for easy understanding and fast diagnosis of structural deterioration by administrators.

The monitoring facility has three technical advantages:

- *Fine-grained:* Modifications of data storage structures, which are hereafter referred by structural changing events, can be captured for each concerned page and the structural deterioration can be estimated in the unit of page. That is, fine-grained monitoring of the structural deterioration is realised, facilitating administrators' identification of the deteriorated portion of the database.
- *Performance-oriented:* The structural deterioration metrics considers physical performance characteristics of the storage device such as hard disk drive. Thus, highly accurate estimation of the structural deterioration is expected.
- *Low overhead:* The sniffer needs to capture a small number of events and the estimator can update the measured structural deterioration in an incremental fashion. Online monitoring of structural deterioration can be realised without fully scanning the database.

The following subsections describe the details of the monitor. First, an estimation method of IO access cost using monitored structure and a calculation method of the degree of structural deterioration are described in §3.2. Next, the exploitation of storage performance characteristics is discussed in §3.3. Finally, an incremental technique for updating the degree of structural deterioration, is described in §3.4. The details of visualiser are described in §4.

3.2 Structural deterioration model

In this section, we introduce a novel model of database structural deterioration. Due to the page limitation, we give discussion with a focus on B+tree, a typical data structure deployed in many database systems. Mathematical symbols used in the paper are summarised in Table 1.

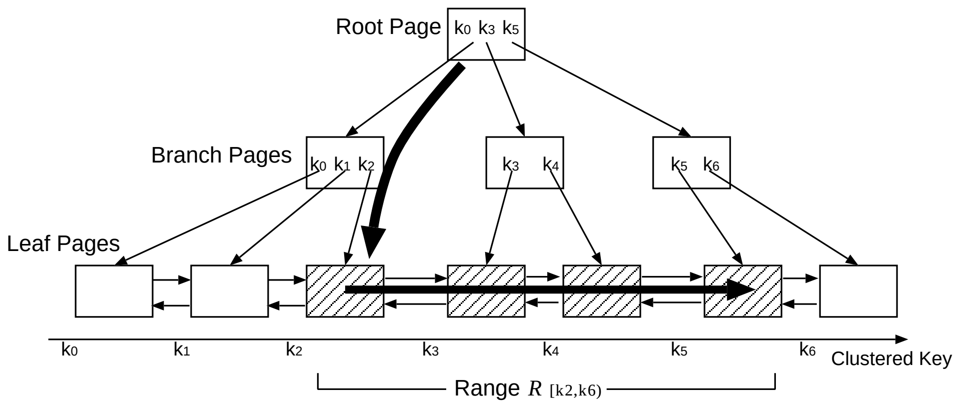
Table 1 Variables, constants and functions

d	Depth of B+tree
$\mathbf{R}$	Cluster key range: $[k_{begin}, k_{end}]$
$C_{\mathbf{R}}$	IO response time of range scan of **R**
$C_{\mathbf{Rv}}$	$C_{\mathbf{R}}$'s element for vertical access
$C_{\mathbf{Rh}}$	$C_{\mathbf{R}}$'s element for horizontal access
$N_{\mathbf{Rh}}$	Number of traversed pages for horizontal access in range **R** scan
p_i	ith leaf page on B+tree $(0 \leq i < N)$
k_i	The smallest cluster key in p_i $k_{i-1} \leq k_i \leq k_{i+1}$
a_i	Address of p_i on the storage
c_i	Expected IO response time of p_i during range scan
c_{seq}	Average IO response time of pages in sequential scan
c_{rnd}	Average IO response time of pages in random scan
$cl()$	Conventional storage performance map
$st()$	Proposed storage performance map

3.2.1 Target structure

Figure 4 illustrates an example of B+tree, which consists of fixed-sized pages and inter-page vertical/horizontal pointers.

Figure 4 B+tree structure of clustered table



Each leaf page is denoted by p_i, where i is a non-negative integer indicating a logical order of the page; a leaf page holding the smallest clustering key is denoted by p_0, a neighbouring page is p_1, and a largest-key page is p_{N-1}. The physical address of each leaf page is denoted by a_i, which is not necessarily equivalent to the logical order i.

3.2.2 Target access pattern

A typical scan operation within the key range $\mathbf{R} = [k_{begin}, k_{end}]$, is conducted in the following two steps.

- *Top-down traverse:* The tree is vertically traversed from the root page to a leaf page p_{i_0} satisfying $k_{i_0} \le k_{begin} < k_{i_0+1}$. The IO cost (response time) of this step is denoted by $C_{\mathbf{Rv}}$.
- *Left-to-right traverse:* From the page $p_{i_0} + 1$, leaf pages are scanned until a page p_{i_1} satisfying $k_{end} < k_{i_1}$ is encountered. The IO cost of this step is denoted by $C_{\mathbf{Rh}}$. The number of pages read in this step is indicated by $N_{\mathbf{Rh}}(= i_1 - i_0)$.

IO cost for the range scan ($C_{\mathbf{R}}$) is represented as $C_{\mathbf{R}} = C_{\mathbf{Rv}} + C_{\mathbf{Rh}}$.

In the top-down traverse, the number of IO is expected to be the depth of B+tree *d*. *d* is $O(logN)$. Pages in upper level are on the database buffer cache more frequently then pages in lower level. In the left-to-right traverse, if **R** is specified, target leaf pages are determined and $N_{\mathbf{Rh}}$ IOs are expected to issued. During a range scan, each leaf page is read only once, so the page is rarely expected to be on the buffer cache if another query accessed the page accidentally in the near past. In fact, for large width of range, $C_{\mathbf{Rv}}$ is very small compared with $C_{\mathbf{Rh}}$. From these considerations, the model need not calculate the impact of database buffer cache on performance while we deal with not so narrow range scan.

3.2.3 IO cost estimation

For simplicity, $C_{\mathbf{Rv}}$ can be approximated to d_{crnd}. Next, we discuss the prediction of $C_{\mathbf{Rh}}$ in details. Range scan operation can be decoupled into a series of leaf page retrievals. We can predict $C_{\mathbf{Rh}}$ in the following formula:

$$C_{\mathbf{Rh}} = \sum_{i=i_0+1}^{i_0+N_{\mathbf{Rh}}-1} c_i$$

where c_i indicates IO response time of a leaf page p_i.

The issue is how to derive c_i. Let us consider the performance characteristics of typical disk drives. When a single read request is issued to a disk drive, the drive actuates its head arm, waits until the target block rotates to the head, retrieves data from the block and transfers the read data. In the above steps, usually the actuation and wait time is dominant for non-sequential access. Interestingly, the actuation time relates to the seek distance, and the wait time is statistically constant. Thus, we assume that the following formula can be used to approximate c_i. (For simplicity, we suppose $c_0 = 0$.)

$$c_i = st\left(a_i - a_{i-1}\right)$$

The map *st*() is described in the next section.

While the distribution $\{c_i\}$ for $0 \le i < N$ is being kept, $C_{\mathbf{Rh}}$ for any range **R** can be estimated.

3.2.4 Structural deterioration degree

On the current data structure and its placement on the secondary storage, IO access cost of range **R** scan is estimated as $C_{\mathbf{R}}$. Then, how much is the cost on the best-organised structure? The answer is that relocating leaf pages so that left-to-right traverse is sequential scan on the secondary storage actually. Since, sequential scan is generally the fastest access on the storage consisted of disk drives. That is, such a structure satisfies the condition $a_i = a_{i-1}+1$ for $i > 0$.

There is no way that top-down traverse is executed sequentially on the secondary storage for arbitrary range **R** even on any data placement because of branching nature of B+tree structure. Therefore, top-down traverse does not mostly affect the performance due to structural deterioration, except for that the depth of B+tree d increases more than necessary. Additionally, the effect can be ignorable compared to that of left-to-right traverse when $N_{\mathbf{Rh}}$ is large enough.

Consequently, IO access cost of range **R** scan on the best-organised structure, $C_{\mathbf{Rh},ideal}$ can be described as follows:

$$C_{\mathbf{Rh},ideal} = \sum_{i=i_0+1}^{i+0+N_{\mathbf{Rh}}-1} c_{seq}$$

$C_{\mathbf{Rv},ideal}$ is the same as $C_{\mathbf{Rv}} = d_{c_{rnd}}$.

Structural deterioration degree SD is calculated as follows:

$$SD = \frac{C_{\mathbf{Rv}} + C_{\mathbf{Rh}}}{C_{\mathbf{Rv},ideal} + C_{\mathbf{Rh},ideal}} = \frac{dc_{rnd} + \sum_{i=i_0+1}^{i_0+N_{\mathbf{Rh}}-1} c_i}{dc_{rnd} + \sum_{i=i_0+1}^{i_0+N_{\mathbf{Rh}}-1} c_{seq}} \approx \begin{cases} \dfrac{\sum_{i=i_0+1}^{i_0+N_{\mathbf{Rh}}-1} c_i / c_{seq}}{N_{\mathbf{Rh}}-1} & (N_{\mathbf{Rh}} \gg 1) \\ 1 & (N_{\mathbf{Rh}} \not\gg 1) \end{cases}$$

Here, c_i/c_{seq} is the fine-grained unit of the degree of structural deterioration. We call $\{c_i/c_{seq}\}$ for $(0 < i < N)$ structural deterioration distribution.

3.3 Storage performance model

This section describes conventional storage performance model, and our proposed storage performance model. Each model is implemented as a mapping table which converts logical seek size *lseek* to expected IO response time. c_i can be derived with the map by input *lssek* as $a_i - a_i - 1$. The internal values of this mapping table are set by issuing indeed several typical access patterns to the secondary storage where database system stores the data.

First, we show a conventional method using *cluster ratio s* (Gassner et al., 1993) to estimate IO access cost. *s* is the ratio of IOs assumed as sequential access in all IOs. The method approximates access cost of each IO as c_{seq} if it is assumed as sequential access, and c_{rnd} if not. The map *cl*() is described as follows:

$$cl(lseek) = \begin{cases} c_{seq} & lseek = 1 \\ c_{rnd} & lseek \neq 1 \end{cases}$$

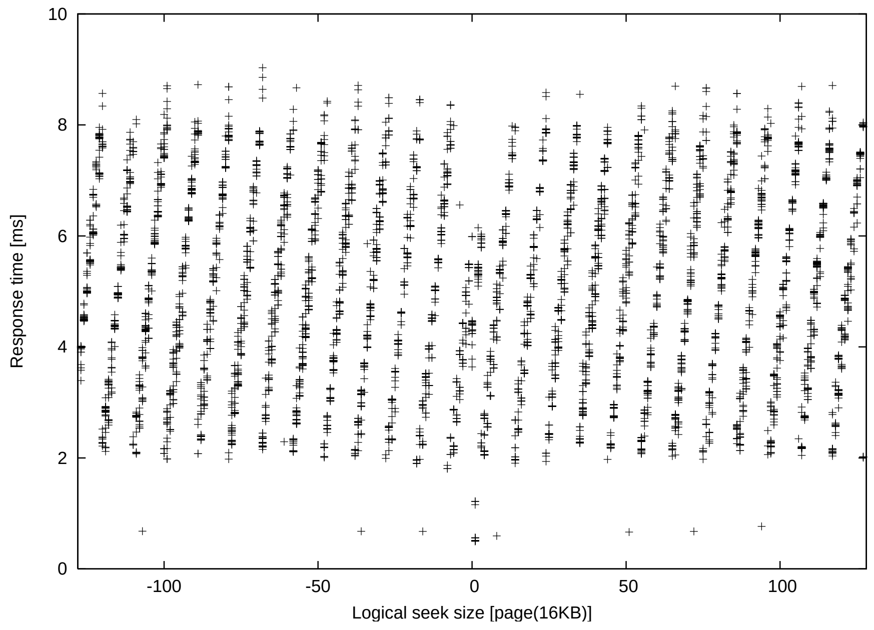
Naively, c_{seq} can be obtained by a sequential scan access, and c_{rnd} can be obtained by a random scan access on the storage.

Next, the details of our proposed storage performance model are described. Typical database systems and file systems have a page allocation strategy that attempts to keep spacial locality as high as possible. For example, page split occurs on leaf pages of a B+tree, newly allocated page is selected on the next address to the old page, or the addresses near the old page prior to other addresses if the space is not be used. On file systems, the pages belonging to a file is tried to be placed nearby physically. Consequently, range scan on B+tree or file traverse has the following characteristics: the number of IOs with smaller *lseek* tend to be larger than the number of IOs with larger lseek, after data modification is repeated and structural deterioration makes progress. The conventional method assumes the cost for non-sequential accesses can be approximated as the cost for random accesses, however, the above fact indicates that the assumption is not true. The actual access cost can be less than the cost estimated by the conventional method.

Considering the strategy, we propose a novel storage performance model; making a detailed map for smaller *lseek*, and a rough map for larger *lseek*. This scheme enables a storage performance map with small footprint, which has the estimation capability with enough accuracy for calculation of the degree of structural deterioration.

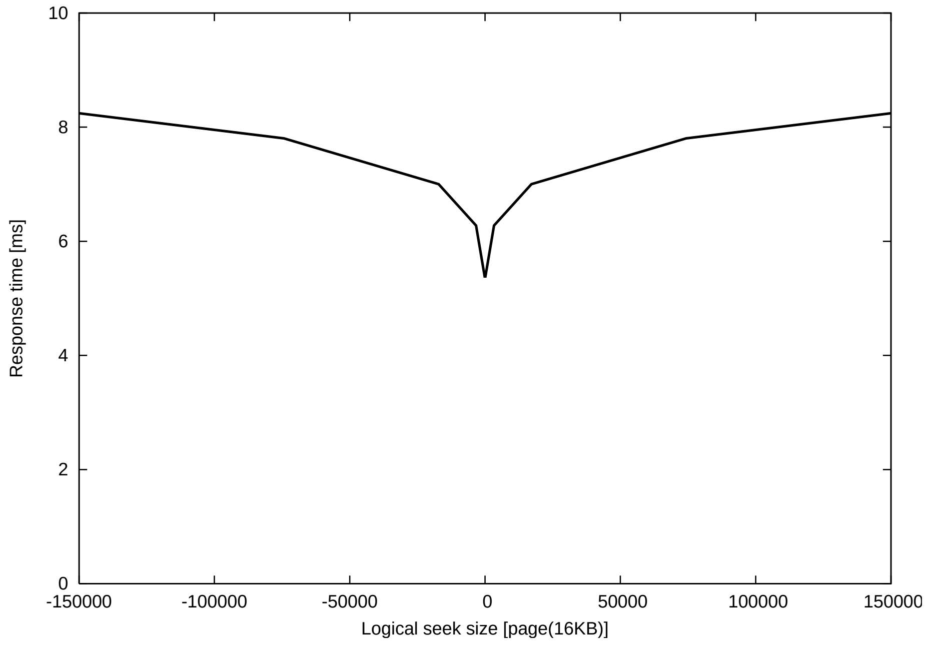
Figures 5 and 6 show a relationship between *lseek* size and response time we measured in executing 16 KB random accesses on a hard disk drive. Figure 5 focuses on the range of small *lseek* sizes where measured points have been plotted and Figure 6 presents the entire range where we have employed linear approximation. In general, disk access time can be decomposed into seek time, rotational wait time and data transfer time. Here the seek time is time duration necessary for the disk head to move onto the target track, the rotational time is for the target sector to rotate to the disk head, and the data transfer time for the disk controller to transfer the requested data. For example, if the disk is accessed in a sequential manner, the response time is determined only the transfer time because the seek time and the rotational wait time can be hidden due to the prefetching. If the access is fully random over the disk space, the access time is mainly subject to the seek size, and the data transfer time is rather negligible. In addition, if the disk access is not fully sequential or random, we should carefully observe the disk behaviour. Based on the observation, we modelled the performance properties of disk drives.

Figure 5 Logical seek size vs. IO response time by random read access on a hard disk drive



Note: Close up to small logical seek area

Figure 6 Logical seek size vs. IO response time by random read access on a hard disk drive



Note: Large logical seek area, linear approximation

The model classifies IO behaviours into three types by *lseek* as follows:

- *Sequential access:* $lseek = 1$. Sequential access on the storage is expected and also large IO is likely to be issued by database system. In this case, head seek time and rotation time inside disk drive are both 0. Consequently, *st*(1) is determined by executing sequential scan with expected large IO. *st*(1) is calculated as (page size)/(measured throughput). In this paper we deal with c_{seq} as *st*(1).
- *Semi-sequential access:* $1 < lseek \leq lseek_{semi}$. Whereas large IO is not issued by database system, sequential prefetch inside disk drive impacts on the IO response time. $lseek_{semi}$ depends on the storage characteristics, it is at most the size of a few tracks of the disk drive. In the case of cache-hit, the cost is the same as that on sequential scan. In other case, head seek time is almost 0 because the *lseek* size is not so large, but rotation time can not be ignored. The internal values of *st*(*lseek*) is determined by executing interval scan where the scan skips $lseek - 1$ pages. We assume the ratio of cache-hit to cache-miss can be approximated as that on the interval scan. Average IO response time of each skipped scan is set to the returned value of *st*(*lseek*).
- *Non-sequential access:* $lseek_{semi} < lseek$, $lseek < 0$. Sequential prefetch does not expected anymore, IO response time of this region depends on head seek time and rotation time in disk drive. Rotation time depends on *lseek* approximately as seen in Figure 5. Consequently, *st*() are determined by executing random scan on the storage and average IO response time for each *lseek* access is set to the returned value *st*(*lseek*). To make the table size be small, *lseek* is enough large ($|lseek| > lseek_{linear}$), we assume the distribution of lseek is uniform in target scan of IO sequence, so the rotation time is assumed to be the time taken by half turn of the disk, the internal values are determined by doing linear approximation of IO response time plot for *lseek* like Figure 6 and the linear function's internal values are set to *st*(*lseek*).

$lseek = 0$ means that the same page are read in a row, the page must be on the database buffer cache at second read, so such IOs are not supposed.

The map function *st*() was derived based on the above classification. Let us present a brief example of our experiment described in §5, where we used a disk drive Cheetah 10 K 18 GB (Seagete, 1999). Here, we assumed that the access block size was set to 16 KB. When the disk was accessed in a sequential fashion (that is, $lseek = 1$), we measured the average response time of each IO and then we could derive $st(1) = 0.584$ ms. In contrast, when the disk was accessed in a semi-sequential fashion (that is, $2 \leq lseek \leq 10$), we also measured the average response time of each IO for respective stride sizes. We could then derive $st(2) = 2.21$ ms, $st(3) = 3.15$ ms, $st(4) = 2.32$ ms, $st(5) = 2.91$ ms, $st(6) = 3.49$ ms, $st(7) = 4.07$ ms, $st(8) = 2.72$ ms, $st(9) = 5.24$ ms and $st(10) = 5.82$ ms. Finally, we divided the case where the disk was accessed in a non-sequential fashion (that is, $lseek > 10$ or $lseek < 0$) into two cases. In the first case ($|lseek| \leq 128$), we directly employed the relationship between *lseek* and response time shown in Figure 5, whereas in the second case ($|lseek| > 128$), we employed the approximated relationship between *lseek* and response time shown in Figure 6. This simple performance model can give reasonable accuracy, which will be studied later in §5.

In order to estimate the impact of target structural deterioration on performance accurately with a simple model, the following assumptions are taken: First, IO sequence is given. Second, the sequence is in units of pages are issued synchronously. Finally, it is ignorable that there is LRU cache in the storage, operating system, and database system, but linear prefetch and coalescence of continuous IOs are allowed in the storage level. We regard such IO sequences as a continuous leaf pages $\left\{p_{i+1},\ldots,p_{i_0+N_{\mathbf{Rh}}-1}\right\}$ of B+tree in the previous section, then left-to-right traverse of range scan in B+tree satisfies above assumptions.

3.4 Incremental maintenance

This section describes incremental update scheme to monitor target structural deterioration with low overhead. Updates of structure are classified into page split and page merge. Other types of updates are not assumed.

Figure 7 shows changes in page addition by split. When a page q is added between $p_i - 1$ and p_i, difference of $C_{\mathbf{Rh}}$ where **R** covers these leaf pages is sum of only differences of IO response time in q and p_i. Figure 8 shows changes in page deletion by merge. When a page p_i is deleted, difference of $C_{\mathbf{Rh}}$ is sum of only differences of IO response time in p_i and $p_i + 1$.

Figure 7 Addition of leaf page to B+tree

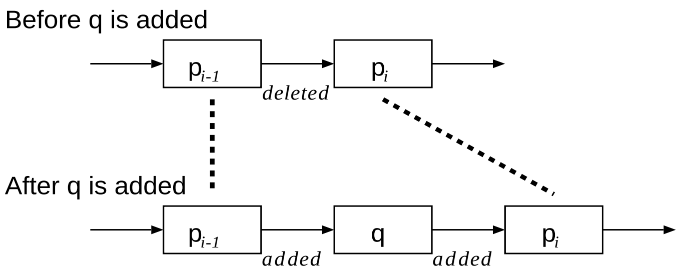
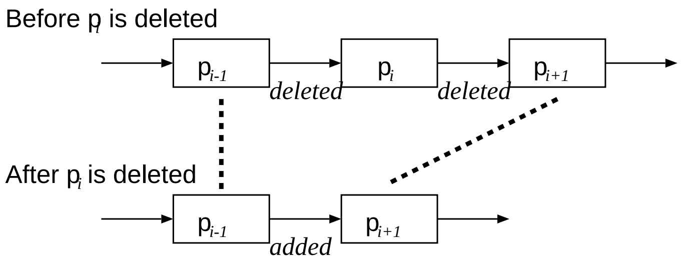


Figure 8 Deletion of leaf page from B+tree



To calculate these differences by using *st*(), information of added (or deleted) page, and also previous and next pages of the page are required; cluster key k and page address a of three pages. Table 2 shows the differences of $C_{\mathbf{Rh}}$ when a page is added or deleted in the position p_{i_0} and also in the case that they occur at edges of leaf pages.

Structural deterioration monitor keeps cluster key k_i and expected IO response time c_i of each leaf page p_i in order to estimate performance of any range in up-to-date state of database structure.

Table 2 Difference of $C_{\mathbf{Rh}}$ by addition or deletion of a page

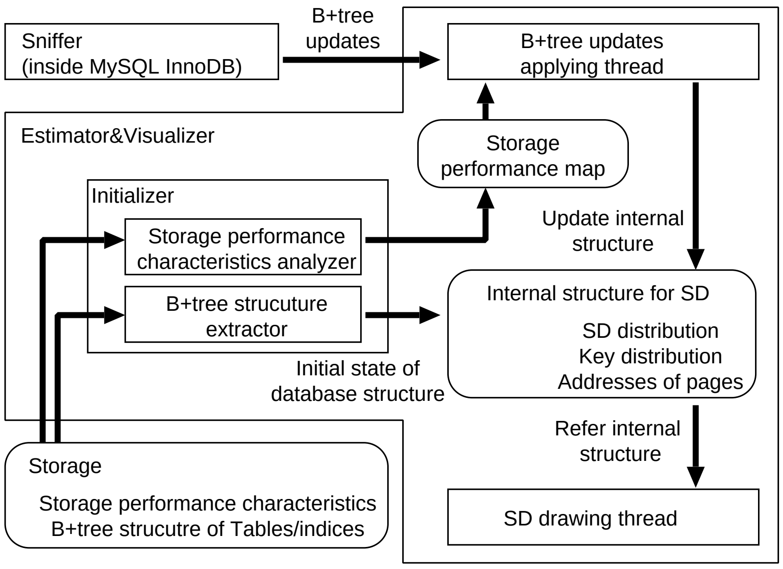
	Add q (address: b) next to p_{i0}	*Delete p_{i0}*
Leftmost	$+st(a_0 - b)$	$-st(a_1 - a_0)$
Rightmost	$+st(b - a_{N-1})$	$-st(a_{N-1} - a_{N-2})$
Other	$-st(a_{i_0} - a_{i_0\text{-}1})$	$-st(a_{i_0} - a_{i_0\text{-}1})$
Position i_0	$+st(b - a_{i_0-1})$	$-st(a_{i_0+1} - a_{i_0})$
	$+st(a_{i_0} - b)$	$+st(a_{i_0+1} - a_{i_0\text{-}1})$

4 Implementation

We implemented a prototype of online structural deterioration monitor. The prototype supports open-source database software MySQL (MySQL AB, 2007) 5.0 InnoDB and it can monitor and visualise the distribution of target structure deterioration of cluster tables or secondary indices constructed by B+tree in almost real time.

The composition of the prototype is shown in Figure 9. The prototype has internal data for structural deterioration and the mapping table *st*() inside. Routines of the prototype consist of mainly three parts: an initialising routine, B+tree update applying thread, and structural deterioration distribution drawing thread.

Figure 9 Design of SD-Mon prototype



The monitor requires updates of database structure, therefore we implemented the sniffer of B+tree update as a small code in MySQL InnoDB engine. The sniffer logs split/merge events of B+tree. The estimator receives the logs via file or pipe and applies it in almost real time.

The estimator and visualiser are implemented collectively as a GUI tool. When the prototype starts up, the initialising routine reads dictionary of the database to obtain schemas of tables and indexes, then it scans B+tree structures of those and create internal data structure for structural deterioration distribution. After initialisation, B+tree update applying thread and structural deterioration distribution drawing thread start. The applying thread is always waiting updates of B+tree structures from the sniffer. The drawing thread draw the structural deterioration distribution when the update events occur or periodically.

The structural deterioration distribution is drawn for two types of x axis; logical axis i (the order of cluster key k_i) and physical axis a_i. Y axis is represented as structural deterioration distribution c_i/c_{seq}. The monitor has colouring function for key range so that administrator can easily understand where the deteriorated data are distributed physically. For these features, the monitor has k_i, c_i, and a_i for each page pi as internal data. If you do not need physical view, a_i need not be kept.

By using updates of the database structure, the monitor can work on another server from the database software. Consequently, the monitor can analyse structural deterioration with enough accuracy and resolution, despite minimal effects on online workload. The monitor applies the updates very fast, so an even large database needs only a cheep server and a cheep storage for the monitor.

5 Evaluation

In this section, the proposed online structural deterioration monitor is evaluated. First, visualisation image of structural deterioration distribution using GUI tool is shown. Next, capability of the monitor is evaluated in terms of accuracy, resolution, and overhead. TPC-H benchmark (TPC, 2007) and a synthetic workload are used for evaluation.

5.1 Experimental setup

The experimental environment is shown in Figure 10. We used PC running RedHat Linux. The PC connects to a JBOD disk array that contains hard disk drives via 1 Gbps Fibre Channel. The JBOD contains Cheetah 10K 18 GB (Seagate, 1999) hard disk drives, where maximum throughput in the most outside zone with sequential read is 28 MB/s measured experimentally. We used two storages for the database; single disk drive and multiple disk drives constructing software RAID0. The software RAID0 storage contains four disk drives and chunk size is 64 KB. The parameters of the storage model for the storages are shown in Table 3. We used each storage as a raw device and allocate 4 GB for database in it. We used MySQL 5.0 InnoDB database engine for whole experiments. Page size was the default 16 KB.

Figure 10 Experimental environment

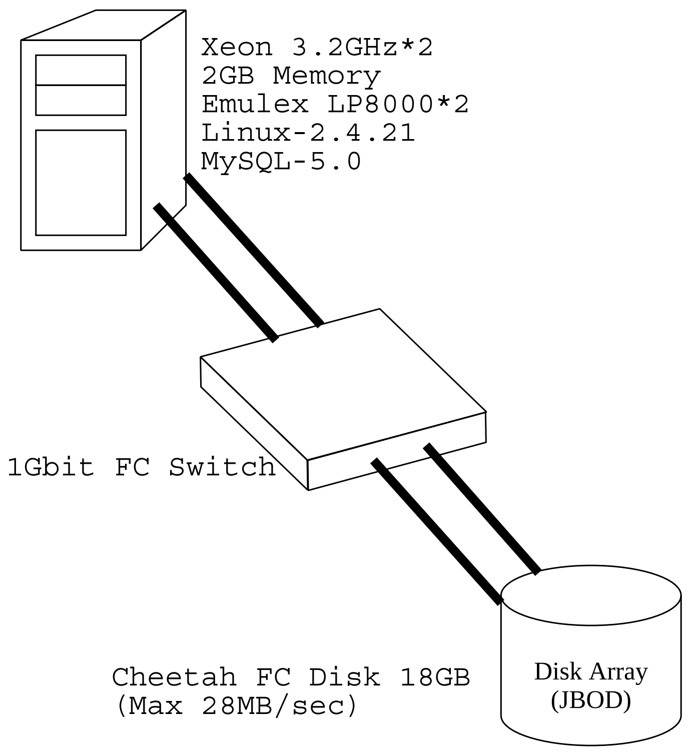


Table 3 Parameters for storage models

Parameter	*For single disk*	*For multiple disks*
c_{seq}	0.584 ms	0.146 ms
c_{rnd}	7.231 ms	6.489 ms
$lseek_{semi}$	10 pages	16 pages
$lseek_{linear}$	128 pages	128 pages

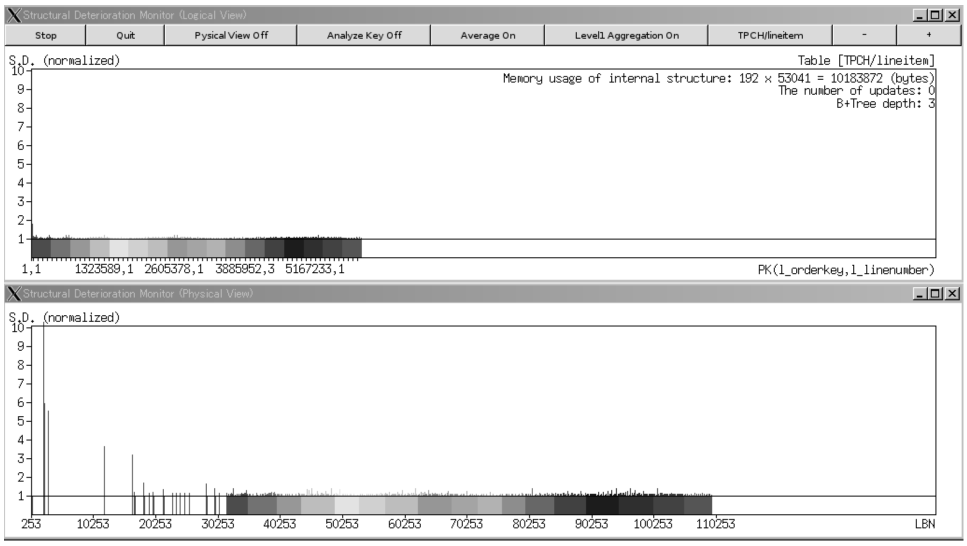
5.2 A case study with GUI tool

Here we show a case study with TPC-H benchmark and prototype GUI tool, which visualises structural deterioration distribution in almost real time.

TPC-H is the standard database benchmark for decision support. We let TPC-H scale factor be SF = 1. We created TPC-H schema using cluster tables. Approximately 1.5 GB of space was used for tables and indexes just after the data was loaded. Space for lineitem table was about 830 MB, and space for orders table was about 190MB. The depth *d* of B+trees of both cluster tables was 3, and did not change during experiment.

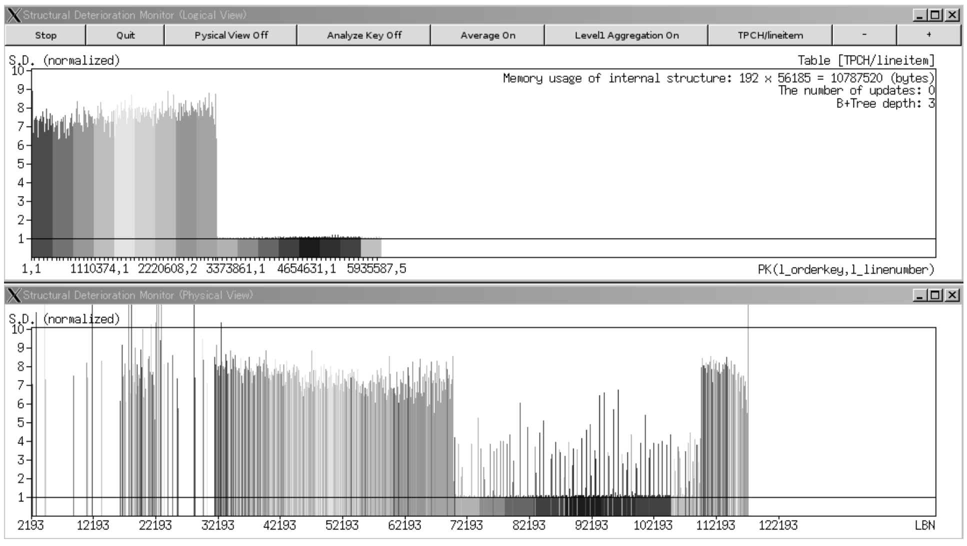
We repeated the following update operations 200 times. In the update operation, refresh query RF1 (bulk insertion of orders table and lineitem table) and RF2 (bulk deletion of orders table and lineitem table) defined by TPC-H benchmark are issued one each. When this update operation is repeated 100 times, all records of orders tables and lineitem tables are updated. 400 times updates makes the data be the same data logically but structural data organisation becomes deteriorated. The update operations change little the amount of records and the distribution of cluster keys in both tables.

Figure 11 Structural deterioration view of lineitem table with SD-Mon prototype. (a) after data load (b) after 50 refreshes (c) after 100 refreshes (d) after 200 refreshes



```
Memory usage of internal structure: 192 x 53041 = 10183872 (bytes)
                                           The number of updates: 0
                                                   B+Tree depth: 3
```

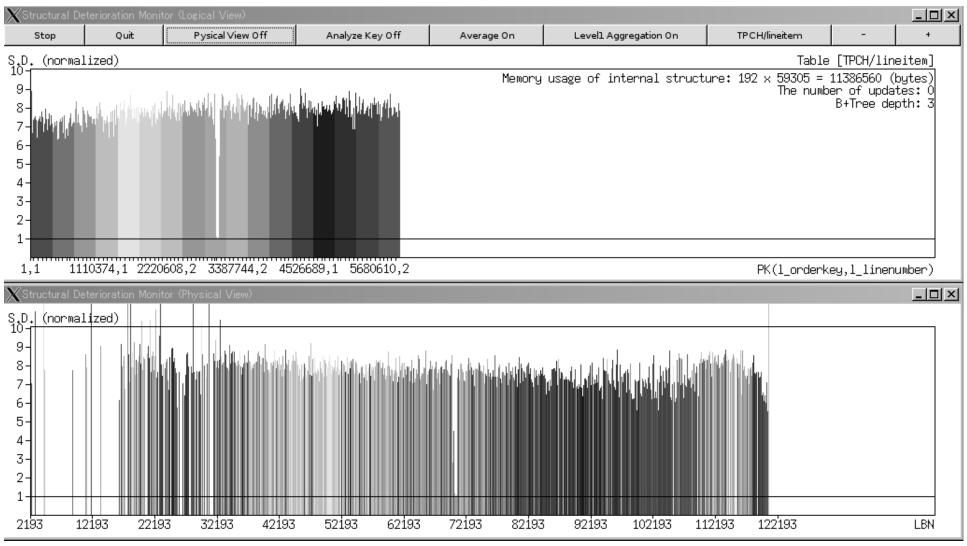
(a)



```
Memory usage of internal structure: 192 x 56185 = 10787520 (bytes)
                                           The number of updates: 0
                                                   B+Tree depth: 3
```

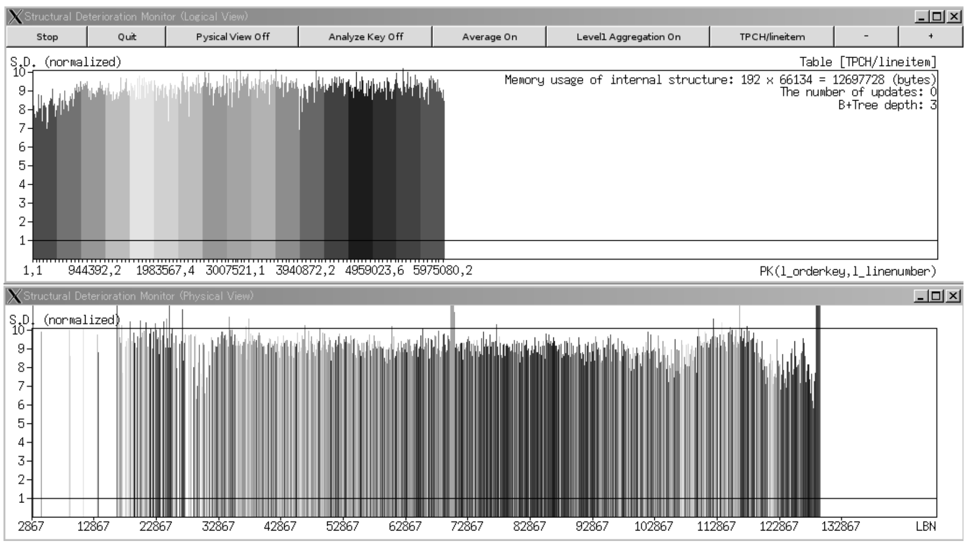
(b)

Figure 11 Structural deterioration view of lineitem table with SD-Mon prototype (a) after data load (b) after 50 refreshes (c) after 100 refreshes (d) after 200 refreshes (continued)



```
Table [TPCH/lineitem]
Memory usage of internal structure: 192 x 59305 = 11386560 (bytes)
The number of updates: 0
B+Tree depth: 3
```

(c)



```
Table [TPCH/lineitem]
Memory usage of internal structure: 192 x 66134 = 12697728 (bytes)
The number of updates: 0
B+Tree depth: 3
```

(d)

First, the changes of structural deterioration distribution of lineitem table of TPC-H drawn by the prototype are shown in Figure 11. The top view is drawn in logical axis, and the bottom view is in physical axis described in §4. At initial state (a), no structural deterioration exists. After 50 refreshes (b), 50% fraction of data are deteriorated by refresh queries. After that, 100% data is deteriorated in (c), and the refresh queries are issued more one cycle for all data in (d). The similar images were made for orders table.

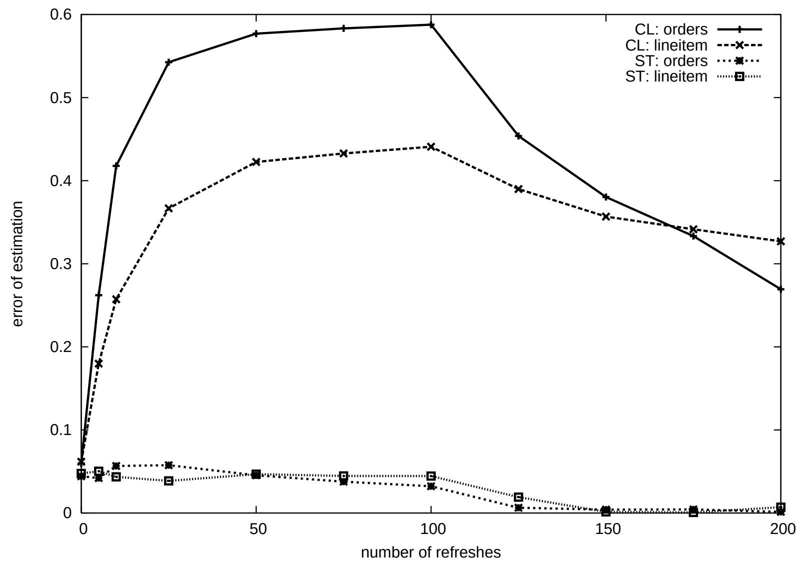
5.3 Model accuracy

Next, we evaluated the accuracy of proposed storage performance model. The database setting and update operations are the same as the previous section. In each interval between update operations, we issued the range scan operations, which scan orders table and lineitem table with full range of cluster key. The response time of them was measured.

Query response time obtained by each scan constructs IO access time to read pages and processing time of the pages. Processing time per each page access is about 0.149 ms in the experimental environment. Thus, measured response time of the range scan was compared with sum of IO response time estimated by the storage model and the time-lag estimated as $((d + N_{\mathbf{Rh}}) \times 0.149$ ms) to evaluate the accuracy of our model.

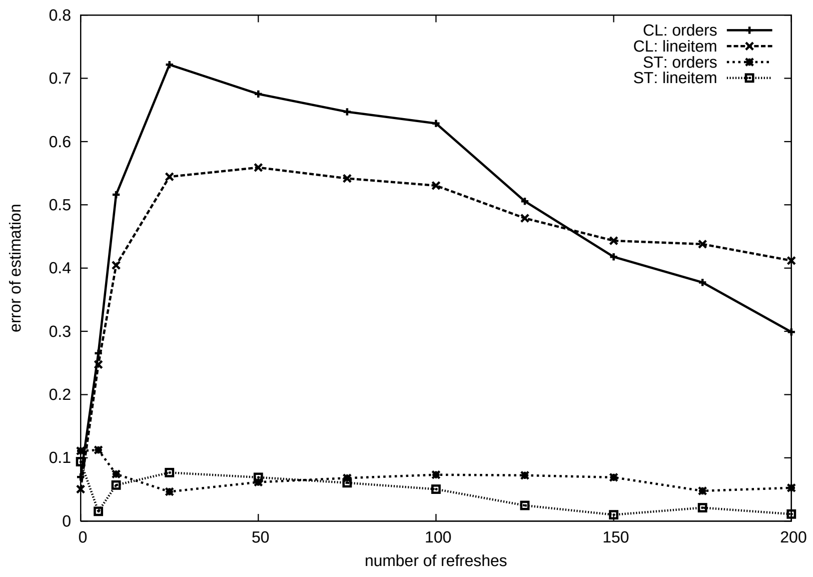
While updates of database are repeated, the response time increases. Figures 12 and 13 show the error of estimation. The error is calculated as estimated response time derived by *cluster ratio* storage model (CL) or our proposed storage model (ST) divided by measured response time. The error of proposed method (ST) is at most 6%, whereas that with cluster ratio is at most 58% for the single disk storage. The result for the multiple disk storage is shown in Figure 13. Comparing with the single disk storage model, the accuracy is a bit less with the multiple disk storage model, it still estimated the response time of range scan in an error of within 8% in most cases, 12% at worst. These results show our new method estimates the performance degradation accurately derived from structural deterioration.

Figure 12 Estimation error of full range scan



Note: Single disk

Figure 13 Estimation error of full range scan



Note: Multiple disks

5.4 Monitoring resolution

Next, we evaluated the resolution of the proposed structural deterioration monitor. Varying key range (scan size), we measured response time of trials of range scan and estimated the IO cost of them with the estimator, as we did in the previous section, on the single disk environment.

Figure 14 Error of estimation in narrow range scan

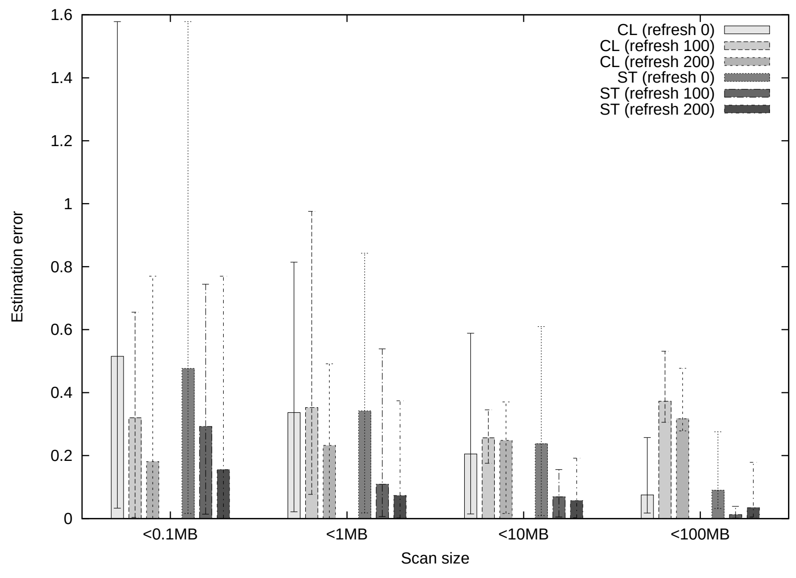


Figure 14 shows estimation errors. As scan size grows, estimation error decreases. For the initial state (refresh 0), larger errors are obtained relatively compared to the other states. In other states, 57% of trials are estimated in an error of within 15% for over 0.1 MB range scan, and 85% of trials are estimated in an error of within 15% for over 1 MB range scan. Comparing two different estimation methods, CL and ST, in almost area, ST estimated with less error than CL.

For smaller range, the assumptions of our proposed method that IO response time can be estimated by average response time for each *lseek* value and that rotation time inside disk drives for large *lseek* value can be approximated as half rotation time are not true. In addition, for the initial state, database system has prefetched too many pages by large IO that includes unnecessary pages for the scan (overprefetch), eventually, the response time has increased. This overhead is within 0.1 sec but not small compared to the total response time for smaller scan size. Top-down traverse did not affect the whole response time because it takes only about 22 msec by almost three IOs and included in the error.

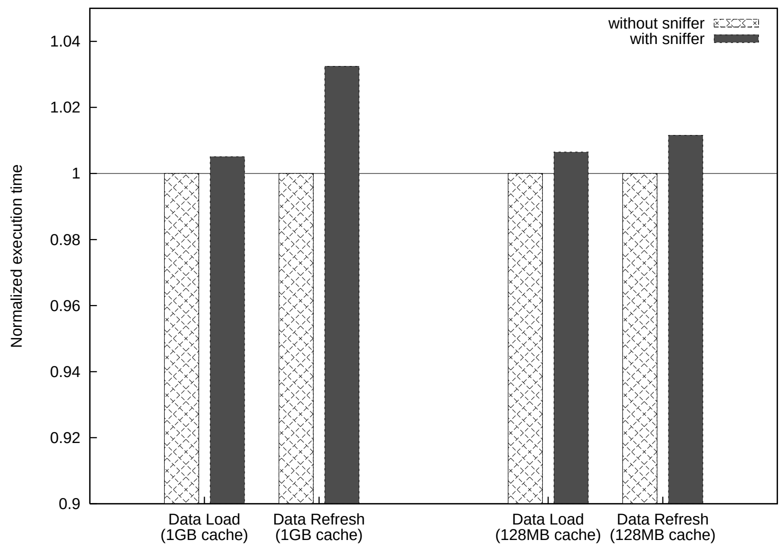
Consequently, when we assume the accuracy of over 1 MB range is enough to believe, the monitor can analyse structural deterioration at 1/1,000,000 resolution for 1 TB database. This show the proposed monitor realises find-grained monitoring of structural deterioration.

5.5 *Monitoring overhead*

Finally, we evaluated the overhead of the sniffer of B+tree update with a foundational experiment using two workload; data load and data refresh. Refresh queries of TPCH has few parallelism and they are not so heavy with our implementation, thus we made a synthetic but heavy workload to a simple cluster table. This table has 600,000 records with distinct cluster keys. Each page can hold about 50 records. After the data was loaded into the table-space, about 1.8 GB was used. Each refresh query consists of three parts; choosing bulk (within 3,000) continuous records from first 40% range of database, delete them from the table, and reinsert them. We issued the refresh query in 70 parallelism, and totally 7,000 times. We believe this workload is so heavy for the monitor because each transaction derive page split or merge almost every time on B+tree structure of the database, totally many structural changes must occur. The execution time of whole updates was measured using MySQL with and without the sniffer, and compared them. We evaluated the overhead varying the size of database buffer cache.

Normalised execution time of data load and data refresh are shown in Figure 15. The base line is the execution time of them without the sniffer. Overheads of the sniffer with 1 GB database buffer cache are about 0.5% for data load, about 3.2% for data refresh. Those with 128 MB database buffer cache are about 0.7% for data load, about 1.2% for data refresh. The hotspot size during data refresh is 40% of 1.8 GB; about 740 MB. With 1 GB buffer cache, whole hot records must be in the cache, thus 3.2% is the upper bound of the overheads of the sniffer in this setting. For large database, the size of database in the secondary storage is much larger than the size of buffer cache in memory, so the case with 128 MB buffer cache of this result is expected in typical situations. Consequently, the proposed scheme can monitor target structural deterioration with low overhead.

Figure 15 Overheads of the sniffer inside DBMS



6 Related work

6.1 Database reorganisation method

Many researches have been investigated on reorganisation method from a long time ago, and some of them are applied to existing commercial products. There are many published papers but we could not describe all of them due to the page limitation.

At the dawn of database system research era, basic reorganisation methods on several data structures and their speeding up techniques were discussed. Sockut and Goldberg (1979) described principles and practices of database reorganisation. Szymanski (1985) proposed a linear algorithm of hash table without additional storage. Omiecinski (1985) and Scheuermann et al. (1989) said record placement problem for the best clustered state is NP hard, and proposed a reorganisation method with several heuristics, then, evaluated the method in subsequent study (Omiecinski et al., 1994) with simulations. On the simulations, they considered relationship between several parameters and recovered performance by reorganisation.

Afterward, many techniques for online reorganisation that reorganisation method and online workload are simultaneously executed, have been mainly researched. Special issue on online reorganisation (Lomet, 1996) bundled several papers regarding online reorganisation. Zou and Salzberg (1996) proposed an online reorganisation method for B+tree structure, where OLTP transactions and small-scale reorganisation are executed simultaneously. Ghandeharizadeh and Kim (1996) proposed two reorganisation methods which relocate data blocks on multiple disk drives in lazy and eager manner. Sockut et al. (1997) combined a fuzzy reorganisation method with a mapping table technique to realise online reorganisation. In this paper, Sockut suggested that the point of time to reorganise should be when:

1 data modification rate is low

2 the applications that require short response time to finish transactions are not working

3 the applications that issues long term transactions are not working.

These suggestions mean that reorganisation workload should be executed in the above conditions, so as not to affect primary workload. Wietrzyk and Orgun (1999) and Wietrzyk et al. (2000) proposed an incremental online reorganisation method for graph data structure. Watanabe and Miura (2002) proposed a reorganisation method which repeats interchanging two physically distant pages.

Several recent researches have considered reorganisation method from a wide view point. Guinepain and Gruenwald (2005) have surveyed data clustering methods, which are a part of data reorganisation methods. They also have described still unsolved research problems; there is not feasible methodology to detect deteriorated portion of database and almost methods substitute some statistics or measured performance; besides, collecting much statistics leads to unacceptable overhead. Ghandeharizadeh et al. (2006) have proposed a reorganisation method, which relocates 'hot' blocks to another disk drive. This method cycles three steps; monitoring access patterns, scheduling block migration, and executing the scheduled migration plan. Whereas this cycle is the same as that we assume, the interval of their cycle is not adaptive.

As described above, many reorganisation methods have been devised. In order to execute these reorganisation method efficiently satisfying SLO policies together, monitoring structural deterioration and scheduling reorganisation by deciding portions and timings are the essential functions.

6.2 Database reorganisation schedule

In terms of database reorganisation schedule, almost studies have focused on the problem that solves the optimal reorganisation interval in order to achieve high cost-efficiency.

Shneiderman (1973, 1974) proposed an analytical method to determine the optimal interval to minimise total cost of reorganisation cost and data access cost, assuming the access cost to data increases as time goes. Das et al. (1975) and Yao et al. (1976) extended the method for the situation where data access cost increases non-linearly. William and Tuel (1978) additionally extend the method for linearly growing files. Lohman and Muckstadt (1977) focused on the amount of data updates and proposed a method to decide batch operation intervals, including reorganisation interval, with an analytical model. Ramírez et al. (1982) boiled down the problem of deciding reorganisation interval to shortest path problem of graph structure, and proposed an algorithm with dynamic programming method. Fung (1984) used instead maximum entropy method.

For specific data structures, analytical methods to determine the point of time of reorganisation have been discussed by Maruyama and Smith (1976), Batory (1982), etc., Chen et al. (1995), Chen and Hassan (1995) and Chen and Banawan (1999) proposed the method that determine optimal reorganisation timing for Bllink-tree structure, assuming uniformly distributed workload. Park and Sridhar (1997) investigated that the optimal timing of reorganisation for VSAM KSDS structure is when region splits start to occur, and proposed an analytical method which estimates when the splits occur.

Many studies focused on predicting the database state in the future with some several assumptions, such as uniformly distributed workload, linearly growing data, constant update rate, etc. Unfortunately, almost such techniques seem to be used for neither commercial database system products nor database administrator practically. This is probably because we could not assume that the workload is uniformly distributed and does not change over time in real database systems. We expect that the detailed information of structural deterioration obtained from our proposed monitor will enhance reorganisation schedule techniques described above and real database systems will use them practically.

6.3 Database monitoring

Studies on database monitoring are described below. Gerlhof et al. (1996) proposed a monitoring scheme which collects detailed statistics of the database system, and a reorganisation method by clustering data utilising the statistics. The overhead of collection method was not so small, at most 34%. Chee et al. (1997) proposed a combination method with reorganisation and prefetch techniques. This study described a monitoring scheme and tried to detect access sequence from data access statistics; it can detect sequential access, interleaved access, and reverse access to each file. Schmidt and Böhlen (2004) proposed a sampling method on B-tree structure to obtain detailed distribution of data. Chaudhuri et al. (2004a) proposed SQLCM: a continuous monitoring framework for relational database engine, which has the capability of low overhead monitoring with an aggregation technique of statistics. Aboulnaga et al. (2004) proposed an autonomic method of understanding distribution of data for query optimiser on a commercial database system product. Chaudhuri et al. (2004b) proposed another method with block-level sampling for the same purpose. Narayanan et al. (2005) proposed a monitoring method to capture various events inside database systems, and predict performance with the obtained information.

Whereas we share the basic concepts with these studies, our purpose and method are different from them. Our purpose is monitoring of database structural deterioration, and we focus on grasping of data structure and estimating IO access cost. Consequently, we succeeded to capture one of most difficult kind of structural deterioration which changes physical IO behaviour from sequential accesses to non-sequential accesses and degrades performance seriously on modern storage devices consisting of hard disk drives. In particular, we achieved it with fine-grained, accurate, and low overhead monitoring technique.

7 Conclusions

The paper proposes a real-time online structural deterioration monitor, an essential function for autonomic database reorganisation. The monitor facilitates structural deterioration analysis and enables efficient reorganisation planning, accordingly relieving the burden of database administrators. We evaluated the monitor can keep track of structural deterioration of changing databases with high accuracy, high resolution, and low overhead. We also developed GUI tool to visualise structural deterioration distribution for database administrators to figure out easily and quickly. We would like to develop a reorganisation scheduling mechanism in future work. We would also like to extend our idea to solid-sate disks (SSDs). Structural deterioration cannot be avoided even in such emerging non-mechanical storage devices. The basic framework of the presented online monitor could be directly applied to SSDs, but we need to modify the model function such that it can efficiently employ the specific performance properties of SSDs.

Acknowledgements

This work has been supported in part by Leading Project 'e-Society' *Advance Storage* and Next-generation IT Program *Development of Out-of-Order Database Engine*, which both are joint research projects between the University of Tokyo and Hitachi Ltd. under the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan. T. Hoshino was partially supported by JSPS fellowship.

References

Aboulnaga, A., Haas, P.J., Lightstone, S., Lohman, G.M, Markl, V., Popivanov I. and Raman, V. (2004) 'Automated statistics collection in db2 udb', *VLDB Conference*, pp.1146–1157.

Batory, D.S. (1982) 'Optimal file designs and reorganization points', *ACM Transactions on Database Systems*, Vol. 7, No. 1, pp.60–81.

Bayer, R. and McCreight, E. (1972) 'Organization and maintenance of large ordered indexes', *Acta Informatica*, Vol. 1, pp.173–189.

Bayer, R. and Schkolnick, M. (1977) 'Concurrency of operations on b-trees', *Acta Informatica*, Vol. 9, pp.1–21.

Chaudhuri, S., Köning, A.C. and Narasayya, V. (2004a) 'SQLCM: a continuous monitoring framework for relational database engines', *IEEE ICDE Conference*, pp.473–484.

Chaudhuri, S., Das, G. and Srivastava, U. (2004b) 'Effective use of block-level sampling in statistics estimation', *ACM SIGMOD Conference*, pp.287–298.

Chee, C.L., Lu, H., Tang, H. and Ramamoorthy, C.V. (1997) 'Improving I/O response times via prefetching and storage system reorganization', *IEEE COMPSAC Conference*, pp.143–148.

Chen, I.R. (1995a) 'A degradable blink-tree with periodic data reorganization', *Computer Journal*, Vol. 38, pp.245–252.

Chen, I.R. and Hassan, S. (1995b) 'Performance analysis of a periodic data reorganization algorithm for concurrent b link-trees in database systems', *ACM Symposium on Applied computing*, pp.40–45.

Chen, I.R. and Banawan, S.A. (1999) 'Performance and stability analysis of multilevel data structures with deferred reorganization', *IEEE Transactions on Software Engineering*, Vol. 25, pp.690–700.

Comer, D. (1979) 'Ubiquitous b-tree', *ACM Computer Survey*, Vol. 11, pp.121–137.

Das, K.S., Teorey, T.J. and Yao, S.B. (1975) 'Reorganization points for file designs with nonlinear processing costs', *VLDB Conference*, pp.516–518.

Fung, K.T. (1984) 'A reorganization model based on the database entropy concept', *Computer Journal*, Vol. 27, No. 1, pp.67–71.

Ganger, G.R., Strunk, J.D. and Klosterman, A.J. (2003) *Self-* Storage: Brick-based Storage with Automated Administration*, Technical Report CMU-CS-03-178, Carnegie Mellon University.

Gassner, P., Lohman, G.M., Schiefer, K.B. and Wang, Y. (1993) 'Query optimization in the IBM db2 family', *IEEE Data Engineering Bulletin*, Vol. 16, No. 4, pp.4–18.

Gerlhof, C.A., Kemper, A. and Moerkotte, G. (1996) 'On the cost of monitoring and reorganization of object bases for clustering', *ACM SIGMOD Record*, Vol. 25, pp.22–27.

Ghandeharizadeh, S., Gao, S., Gahagan, C. and Krauss, R. (2006) 'An on-line reorganization framework for san file systems', *ADBIS Conference*.

Ghandeharizadeh, S. and Kim, D. (1996) 'On-line reorganization of data in scalable continuous media servers', *DEXA Conference*, pp.751–768.

Guinepain, S. and Gruenwald, L. (2005) 'Research issues in automatic database clustering', *ACM SIGMOD Record*, Vol. 34, No. 1, pp.33–38.

Kephart, J.O. and Chess, D.M. (2003) 'The vision of autonomic computing', *IEEE Computer*, Vol. 36, No. 1, pp.41–50.

Lehman, P.L. and Yao, S.B. (1981) 'Efficient locking for concurrent operations on b-trees', *ACM Transactions on Database Systems*, Vol. 6, pp.650–670.

Lightstone, S., Schiefer, B., Zillo, D. and Kleewein, J. (2003) 'Autonomic computing for relational databases: the ten year vision', *AUCOPA Workshop*.

Lightstone, S., Lohman, G.M. and Zilio, D. (2002) 'Toward autonomic computing with db2 universal database', *ACM SIGMOD Record*, Vol. 31, No. 3, pp.55–61.

Lohman, G.M. and Muckstadt, J.A. (1977) 'Optimal policy for batch operations: backup, checkpointing, reorganization, and updating', *ACM Transactions on Database Systems*, Vol. 2, No. 3, pp.209–222.

Lomet, D. (Ed.) (1996) 'Special issue on online reorganization', *IEEE Data Engineering Bulletin*, Vol. 19, No. 2.

Mackert, L.F. and Lohman, G.M. (1989) 'Index scans using a finite LRU buffer: a validated I/O model', *ACM Transactions on Database Systems*, Vol. 14, No. 3, pp.401–424.

Markl, V., Lohman, G.M. and Raman, V. (2003) 'LEO: an autonomic query optimizer for db2', *IBM Systems Journal*, Vol. 42, No. 1, pp.98–106.

Maruyama, K. and Smith, S.E. (1976) 'Optimal reorganization of distributed space disk files', *Communications of the ACM*, Vol. 19, No. 11, pp.634–642.

MySQL: The World's Most Popular Open Source Database, available at http://www.mysql.com/.

Narayanan, D., Thereska, E. and Ailamaki, A. (2005) 'Continuous resource monitoring for self-predicting dbms', *IEEE MASCOTS Conference*, pp.239–248.

Omiecinski, E., Lee, L. and Scheuermann, P. (1994) 'Performance analysis of a concurrent file reorganization algorithm for record clustering', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 6, No. 2, pp.248–257.

Omiecinski, E. (1985) 'Incremental file reorganization schemes', *VLDB Conference*, pp.346–357.

Park, J.S. and Sridhar, V. (1997) 'Probabilistic model and optimal reorganization of b$^+$-tree with physical clustering', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9, No. 5, pp.826–832.

Ramírez, R.J., Tompa, F.W. and Munro, J.I. (1982) 'Optimum reorganization points for arbitrary database costs', *Acta Informatica*, Vol. 18, pp.17–30.

Scheuermann, P., Park, Y.C. and Omiecinski, E. (1989) 'Heuristic reorganization of clustered files', *FODO Conference*, pp.16–30.

Schmidt, A. and Böhlen, M.H. (2004) 'Parameter estimation using b-trees', *IEEE IDEAS Conference*, pp.325–333.

Seagate (1999) *Cheetah 18LP FC Disk Drive Product Manual Volume 1*.

Shneiderman, B. (1973) 'Optimum data base reorganization points', *Communication of the ACM*, Vol. 16, No. 6, pp.362–365.

Shneiderman, B. (1974) 'Opportunities for data base reorganization', *ACM SIGMOD Record*, Vol. 6, pp.1–8.

Sockut, G.H., Beavin, T.A. and Chang, C.C. (1997) 'A method for on-line reorganization of a database', *IBM Systems Journal*, Vol. 36, pp.411–436.

Sockut, G.H. and Goldberg, R.P. (1979) 'Database reorganization – principles and practice', *ACM Computer Survey*, Vol. 11, No. 4, pp.371–395.

Swami, A.N. and Schiefer, K.B. (1995) 'Estimating page fetches for index scans with finite LRU buffers', *VLDB Journal*, Vol. 4, No. 4, pp.675–701.

Szymanski, T.G. (1985) 'Hash table reorganization', *Journal of Algorithms*, Vol. 6, No. 3, pp.322–335.

Telford, R., Horman, R., Lightstone, S., Markov, N., O'Connell, S. and Lohman, G.M. (2003) 'Usability and design considerations for an autonomic relational database management system', *IBM Systems Journal*, Vol. 42, No. 4, pp.568–581.

TPC: Transaction Processing Performance Council, available at http://www.tpc.org/.

Uttamchandani, S., Voruganti, K., Srinivasan, S., Palmer, J. and Pease, D. (2004) 'Polus: growing storage QoS management beyond a 'four-year old kid'', *USENIX FAST Conference*, pp.31–44.

Wang, M., Au, K., Ailamaki, A., Brockwell, A., Faloutsos, C. and Ganger, G.R. (2004) 'Storage device performance prediction with cart models', *ACM SIGMETRICS Conference*, pp.412–413.

Watanabe, S. and Miura, T. (2002) 'Reordering B-tree files', *ACM SAC Symposium*, pp.681–686.

Wietrzyk, V.S. and Orgun, M.A. (1999) 'Dynamic reorganization of object databases', *IEEE IDEAS Conference*, pp.110–118.

Wietrzyk, V.S., Orgun, M.A. and Varadharajan, V. (2000) 'On the analysis of on-line database reorganization', *ADBIS Conference*, pp.293–306.

William, J. and Tuel, G. (1978) 'Optimum reorganization points for linearly growing files', *ACM Transactions on Database Systems*, Vol. 3, pp.32–40.

Yao, S.B., Das, K.S. and Teorey, T.J. (1976) 'A dynamic database reorganization algorithm', *ACM Transactions on Database Systems*, Vol. 1, No. 2, pp.159–174.

Zilio, D.C., Lightstone, S. and Lohman, G.M. (2003) 'Trends in automating physical db design', *IEEE INDIN Conference*, pp.441–445.

Zilio, D.C., Rao, J., Lightstone, S., Lohman, G.M., Storm, A., Garcia-Arellano, C. and Fadden, S. (2004) 'DB2 design advisor: Integrated automatic physical database design', *VLDB Conference*, pp.1087–1097.

Zou, C. and Salzberg, B. (1996) 'On-line reorganization of sparsely-populated B^+-trees', *ACM SIGMOD Conference*, pp.115–124.