A Study on OLTP Performance Degradation by Structural Deterioration of Database

Takashi Hoshino
Graduate School of Information Science and Technology,
University of Tokyo
7-3-1 Hongo, Bunkyo-ku,
Tokyo, 113-0033, Japan
hoshino@tkl.iis.u-tokyo.ac.jp

Kazuo Goda and Masaru Kitsuregawa
Institute of Industrial Science,
University of Tokyo
4-6-1 Komaba, Meguro-ku,
Tokyo, 153-8505, Japan
{kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract

Database updates disorganize data stored physically in secondary storage, which is called structural deterioration and causes performance degradation. Online Transaction Processing(OLTP) is an essential data processing scheme for e-finance, e-commerce and so on. Conventional researches to improve OLTP performance lack a view from the aspect of preventing performance degradation due to structural deterioration. Recently, more and more business operations and services are being digitized then 24-hours-a-day operation of database is required. Needs for techniques to monitor structural deterioration and prevent performance degradation are increasing more and more. In this paper, we analyze how structural deterioration effects performance degradation in OLTP workload by using a benchmark. We also consider strategies of database mutations which are adaptive to workload and prevent structural deterioration from being accelerated.

1 Introduction

Online Transaction Processing (OLTP) is an essential data processing scheme to guarantee ACID properties of transactions for e-businesses such as e-finance, e-commerce. Thus, higher performance of OLTP is required still now. Besides, technologies to realize full-time database operations are required because of rapid digitization of recent business operations and services.

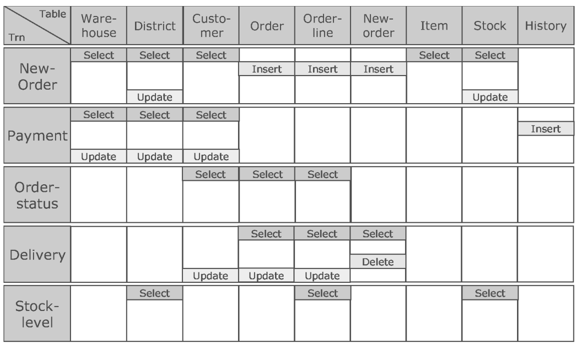
There are two aspects of OLTP speeding up: one is improvement of database structures and processing methods to get higher performance, and the other is prevention of performance degradation due to structural deterioration which is the phenomenon that database structure becomes inefficient for accesses gradually or suddenly caused by accumulated database updates.

From the former aspect, so far, many works to process online transactions faster were researched: some of them tried to optimize structure of database inside secondary storage[1, 2], another one is aware of CPU instructions and cache[3]. Parallelization[4] from various aspects, automatic configuration and tuning of hardware composition or database schema adjusting to workload characteristics [5, 6] were also explored.

From the latter aspect, to remove or slow structural deterioration is also essential approach, since much data mutations involving record inserts, deletes, and updates are executed continuously in OLTP workloads in general. In order to remove structural deterioration, database administrator executes database reorganization that relocates data in the secondary storage to recover performance. Database reorganization is a heavy task that tends to occupy system resources, especially IO resources of the secondary storage which is likely to be a bottleneck of the system. Therefore, online database reorganization methods[7, 8] were researched to deconcentrate IO load of the reorganization or to execute the reorganization in background not to effect on foreground workloads as much as possible. However, the methods to slow structural deterioration, which are especially well-adapted to various workloads dynamically, are not still researched well.

In this paper, we analyze performance degradation of a typical OLTP workload due to structural deterioration with an experimental result by using a benchmark, and discuss strategies which are adaptive to data access patterns to slow structural deterioration. The strategies contributes to prevention of OLTP performance from going down, and also to reduction of the overall resource utilization for database reorganization.

The rest of the paper is organized as follows: Section 2 describes the relationship between OLTP workload and structural deterioration. Next, we analyze performance degradation of OLTP workload in Section 3. In Section 4, we discuss strategies to slow structural deterioration. Related works are described in Section 5, and finally we con-



Trn \ Table	Ware-house	District	Custo-mer	Order	Order-line	New-order	Item	Stock	History
New-Order	Select	Select, Update	Select	Insert	Insert	Insert	Select	Select, Update	
Payment	Select, Update	Select, Update	Select, Update						Insert
Order-status			Select	Select	Select				
Delivery			Update	Select, Update	Select, Update	Select, Delete			
Stock-level		Select			Select			Select	

Figure 1. TPC-C tables and transactions

clude and describe future work in Section 6.

2 OLTP and Structural Deterioration

In this section, we describe characteristics of OLTP workload, potential types of structural deteriorations, and their impacts on OLTP performance.

2.1 Characteristics of OLTP Workload

To discuss relationships between OLTP workload and structural deterioration in general fashion is not so easy, because characteristics of structural deterioration depends on database structures and access patterns. In this paper we analyze them with specific OLTP workload and database system.

TPC-C benchmark[9] is the standard benchmark for OLTP workload, which simulates the activity of a wholesale supplier, and widely used today. There are five transactions and nine tables defined in the specification. Figure 1 shows each transaction accesses to which tables and how. Data access patterns are not always approximated as random accesses. There are also sequences of transactions which access certain tables in order of primary key, which we call *sequential record access*. For example, records of *order*, *order-line*, and *new-order* table are inserted and selected in order of order ID (*o_id*), which is a part of composite primary key in the tables. Locality of accesses tends to be high in such tables. For example, while the records in *order* tables corresponding to the orders waiting to be delivered are accessed frequently, the records corresponding to the delivered orders would not almost be accessed. Above characteristics are expected to be true in other OLTP workloads in the world.

2.2 Structural Deterioration of Database

Structural deterioration is the phenomenon that database structure is mutated, which causes performance degradation by increasing the number of IO or response time per IO for certain access patterns.

Several examples of structural deteriorations are: disordering of pages, decrease of fill factor, fragmentation of records, increase of the depth of B+tree, invalidation of high-water mark, depletion of unallocated pages, and so on. Disordering of pages and decrease of fill factor are common structural deteriorations in many database systems. As for structure deteriorations listed above except for disordering of pages, the number of IO is the important factor for performance degradation. Disordering of pages and decreasing of fill factor are discussed in [10]. Disordering of pages unlikely effects on OLTP performance. Since it decreases performance of range scan especially for large amount of records, which seldom occurs in OLTP workload, and such access patterns do not defined in TPC-C specification. Thus, we focus on decreasing of fill factor and its effect on performance of OLTP workload.

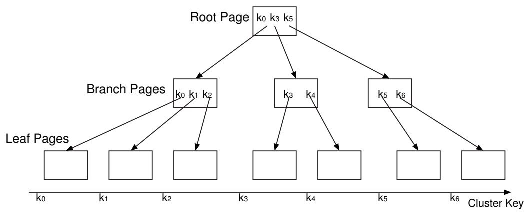


Figure 2. B+tree structure of cluster table

3 Analysis of OLTP Performance Degradation caused by Structural Deterioration

In this section, we analyze performance degradation of transactions caused by database structural deterioration through experiments.

3.1 Target Database Structure

We used B+tree structure which typically constructs cluster table or secondary index of relational database. The structure is adopted as cluster table in MySQL[11] and also as index-organized table in Oracle. In the following part of the paper, MySQL InnoDB database engine is used for experiments. Figure 2 shows the structure of the target B+tree. The page size is fixed. Data records are stored in leaf pages only and in order of cluster key logically for arbitrary records or range to be accessed fast by specifying cluster keys.

Page splits and merges occur by insert, update, and delete of records. In general database systems, target fill factor at time of data load, and threshold fill factor to be merged are tunable, however, in MySQL InnoDB, data records are loaded into leaf pages in the fill factor of 15/16, and pages are merged if possible after a certain period of time records are deleted. Record fragmentation in updates does not be supported, instead, deletion and insertion are used alternatively. Page splits are executed when an insert or update will overflow the page. There are two cases: In one case when the cluster key of inserted record just before the insert is the max in the page, and the cluster key of the record to be inserted are larger than the previous one, InnoDB deems the access pattern is *sequential insert*, then the page are split at the rate of 15/16:1/16. In other cases, the page are split by half-and-half. InnoDB engine distinguishes inserts on data load from normal inserts in the way described above.

3.2 Environment

The experimental environment is described below. We used PC running RedHat Linux, which consists of two Xeon 3.2GHz CPUs and 2GB Memory. The PC connects to a JBOD disk array that contains Cheetah 10K 18GB [12] hard disk drives via 1Gbps Fibre Channel. We used software RAID0 which contained four disk drives and chunk size of which was 64KB. We used the storage as a raw device and allocated 4GB for database in it. We used MySQL 5.0 InnoDB database engine and implemented an additional feature on the MySQL software in order to monitor IOPS (IO per second) of each table/index for analysis. Page size was the default 16KB.

3.3 Settings

We used TPC-C Rev. 5.6 for OLTP benchmark. All tables were cluster tables where each primary key were set as composite cluster key. All variable-length strings were dealt with as fixed-length strings, then all records were fixed-length in the experiments. The number of warehouses was 16. Transaction mix was: *new-order*: 45%, *payment*: 45%, *order-status*: 2%, *delivery*: 6%, *stock-level*: 2%. After data was loaded, 100,000 transactions were executed in five processes per warehouse, in 80 parallel totally, with the think time of 0. Response time of transactions and distribution of fill factor were measured.

3.4 Results

In this experiment, average throughput of transactions is 1138 tpm (transactions per minute). Figure 3 shows the response time of all transactions varying with time. We see that the response time of *new-order*, *payment*, *order-status*, and *delivery* increases from 2,000 elapsed second to 2,500 elapsed second approximately.

Response time break down of each transaction are shown in Figure 4–8. *Phase I* of *new-order* transaction in Figure 4 consists of *select warehouse*, *select customer*, *select district*, and *update district*. *Phase II* consists of *select item*, *select stock*, *update stock*, and *insert order-line*. In similar way, *Phase I* of *payment* transaction in Figure 5 consists of

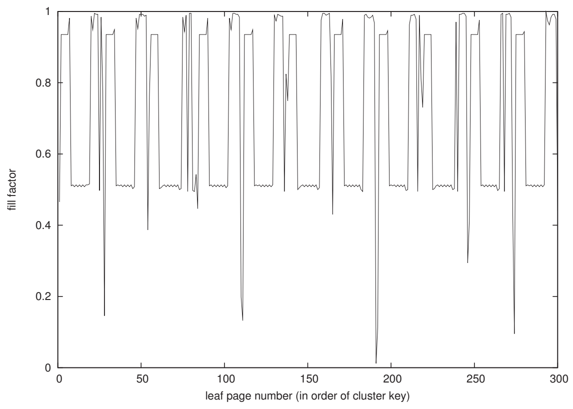


Figure 9. Fill factor of each leaf page (in order of cluster key) in *order* table after executing transactions

update warehouse, *select warehouse*, *update district*, and *select district*. *Phase II* consists of *select customer*. The waiting time on locks is dominant in *Phase I* of both transactions. Other break downs show the access time of *order* table increases especially with time.

Figure 9 shows fill factor of each leaf page in *order* table in order of cluster key after executing all transactions. Cluster key of *order* table is the composite key that consists of warehouse ID (*w_id*), district ID (*d_id*), and order ID (*o_id*). In the initial data load phase, 3,000 records per warehouse per district, *o_id*s of which are from 1 to 3,000 distinctly, are loaded to *order* table at the fill factor of 15/16. In the transaction phase, each *new-order* transaction inserts a record into *order* table. *O_id* of the record is 1 plus that of just previously inserted record, starting from 3,001 for each pair of warehouse and district respectively. When a page split occurs in this phase, the page is split into two pages which fill factors are both 1/2 approximately. The insert into the left page does not occur after the split, and the sequential inserts continue in the right page. After that, the records in the left page are updated but the size does not changed; Neither delete nor more inserts occur. After this access pattern is repeated, the split pages in *order* table by transactions are at the fill factor of 1/2 approximately.

Successive transactions access the records in *order* table in order of *o_id*. The first *delivery* transaction accesses the record with *o_id* of 2,101. That is, first 900 times of *delivery* transactions access the records which are loaded initially, after that, they access the records which are inserted by the *new-order* transactions. Thus, required number of IOs per *delivery* transaction on *order* table in the latter situation, should be approximately twice of that in the former situation. We considered the phenomenon causes increase of response time of the transactions that access *order* table.

Next, we changed the transaction mix, removing *order-*

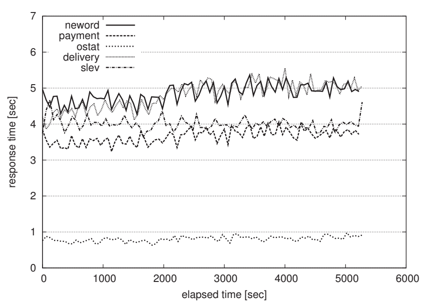


Figure 3. Response time of TPC-C transactions

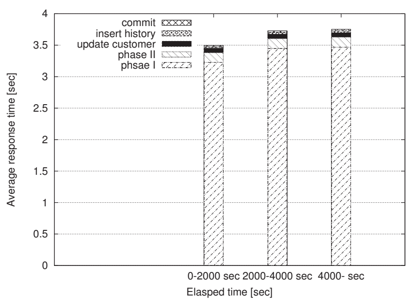


Figure 5. Breakdown of *payment* **transaction**

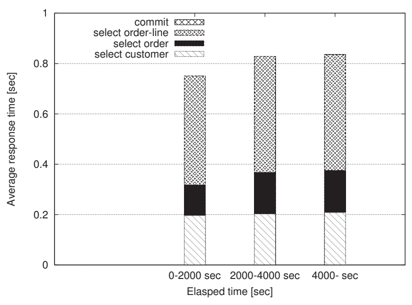


Figure 7. Breakdown of *order-status* **transaction**

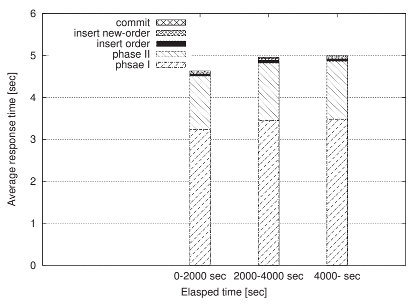


Figure 4. Breakdown of *new-order* **transaction**

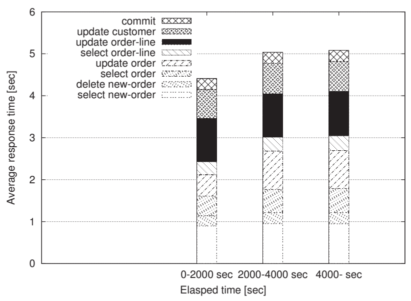


Figure 6. Breakdown of *delivery* **transaction**

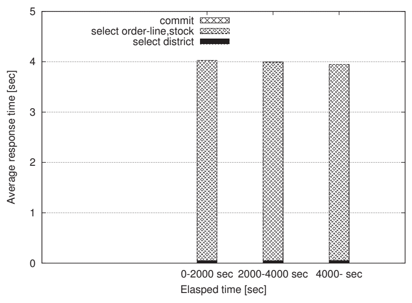


Figure 8. Breakdown of *stock-level* **transaction**

status and *stock-level* that do not change any record: *new-order*: 47%, *payment*: 47%, *delivery*: 6%. Transactions were executed in several buffer cache size settings varied from 16MB to 128MB, and IOPS were measured for each table/index. The database before the transaction phase started in all experiments was the completely same image as the one in the previous experiment. Figure 10–13 show IOPS in the transaction phase. Average transaction throughput of each buffer setting is 1,241, 1,479, 2,018, 3,030 tpm respectively.

`Orders:PRI` in Figures show the IOPS of *order* table. In 16MB buffer cache setting in Figure 10, IOPS starts at about 20, increasing dramatically in 1,500 elapsed second approximately, and keeping about 35 after that. IOPS on other tables and indexes are a bit decreased, because the phenomenon that IOPS on *order* table was increased decreases the throughput of whole transactions with decreasing IOPS. Response time of transactions are related to each other closely due to concurrent resource accesses with locks.

IOPS of *order* table are decreased in more than 64MB buffer cache size setting. Especially, it is ignorable that the impact of increase of IOPS in 128MB setting. We infer that the records accessed by *delivery* transactions are still on cache after some period of time they are inserted by *new-order* transactions with large buffer cache. IOPS does not increase in such a situation. In other experiment with more than 192MB buffer cache, IOPS in *order* table was approximately 0, almost all of hotspot data pages[1] are cached in main memory.

From above consideration, while hotspot data area can settle completely on the buffer cache, decrease of fill factor may not effect on performance degradation of transaction directly. However, if the size of buffer cache is not so large that all hotspot data cannot be cached, fill factor has large impact. Decreasing of fill factor enlarges hotspot size, thus it is still waste of main memory even if there is large buffer cache. To prevent performance degradation of transactions, keeping high fill factor is a good strategy.

The access pattern on *customer*, *stock*, and *item* table are approximated as random record accesses and hotspots of them are large, almost entire tables. Buffer cache hit ratio on the tables are lower than those on other tables, thus IOPS of them increase gradually as increasing buffer cache size and throughput of transactions.

4 Discussion

In this section, we consider strategies of database mutation to slow structural deterioration, adjusting to access pat-

[1]In this case, the leaf pages in *order* table where the records have been inserted by *new-order* transactions, which still does not delivered by *delivery* transactions

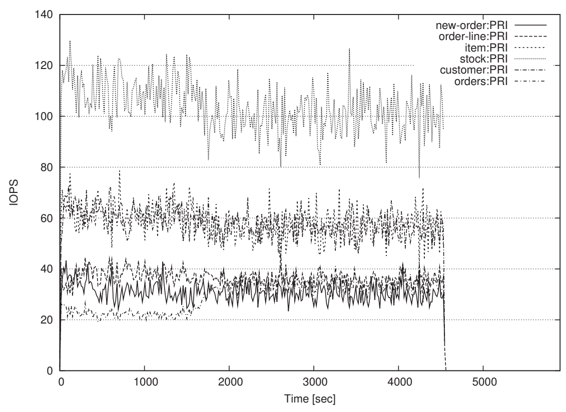


Figure 10. IOPS (16MB buffer cache)

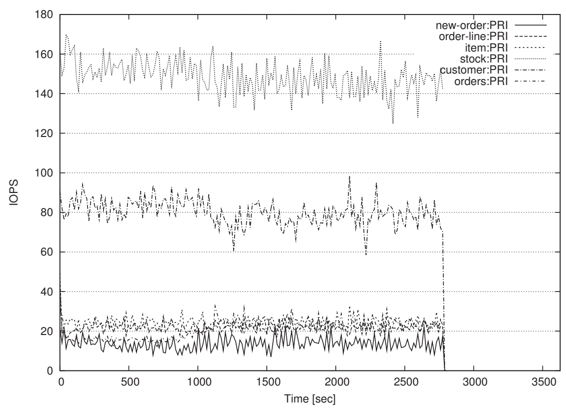


Figure 12. IOPS (64MB buffer cache)

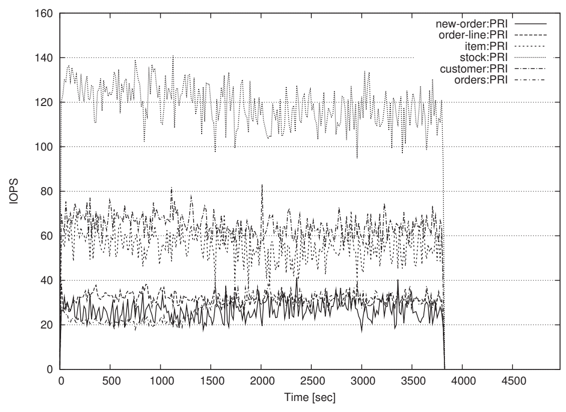


Figure 11. IOPS (32MB buffer cache)

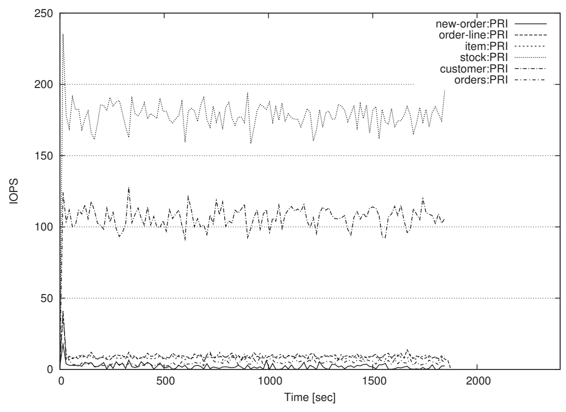


Figure 13. IOPS (128MB buffer cache)

terns which appear in OLTP workload analyzed in section 3 and similar ones.

Even if the system has small buffer cache size and decrease of fill factor may impact on performance of transactions, the structural deterioration can be deferred by rising fill factor at page split.

MySQL 5.0 InnoDB engine detects sequential inserts at each page. The detection method works effectively at the end of table only, where the record to be inserted has the largest cluster key in the table. In this analysis, the composite key was defined in *order* table and sequential inserts occurred intermediately, the detection method did not work well, then fill factor became about a half and performance of transactions were degraded.

We can handling this issue by improving the method that detects sequential insert. We assume there is a record to be inserted into a page, and there is also the record just previously inserted into the page. If two records are adjacent, where there is no record in the page whose cluster key is between the cluster keys of the records, the system regards the inserts as sequential inserts and predicts that near-future inserts into the page are also continuous sequential inserts. The cluster key of the record inserted lastly must be stored at each page for this functionality. It can be implemented by storing the offset of inserted record in the page header.

When the sequential insert overflows the page, the page must be split. If the sequential insert is ascending order and there are (1) records in the page whose cluster keys are larger than the record to be inserted, the records must be stored in the right page by split. The left page has the inserted record and (2) records whose cluster keys are smaller than that of the inserted record. If there do not exist such records described as (1), the page can be split at any fill factor. When the sequential insert is the descending order, corresponding operations must be executed.

This method can detect sequential insert at any position correctly and has a opportunity to tune the fill factor of each page to be high adjusting access patterns, which are represented as the throughput of inserts, deletes, and updates in each page or each range, and the rate of sequential accesses of them, etc. This has effect of not only decreasing the number of page split but also keeping performance of transactions to be high. For example in this analysis, only sequential inserts change the amount of data of *order* table, thus 100% fill factor can be applied when the splits can be allowed with any fill factor in the situation satisfying above conditions.

5 Related Work

Many works such as [7, 13, 8, 14, 10] considered structural deterioration of database and reorganization. These works mainly focus on monitoring of structural deterioration and method of database reorganization to execute fast in the background without effect on foreground workload. This paper considers the strategies slowing structural deterioration in mutation of database structures, which is complementary approach with making database reorganization method be sophisticated.

Johnson et al.[1] presented the fact that the merge of pages does not required where data population is growing, assuming random access to B+trees. This paper considered split strategies to reduce the number of splits and keep fill factor to be high.

Lomet [2] proposed a method which provides highly concurrent access to B+trees. To almost of variants of B+trees like this, we expect our idea can be applied.

For database monitoring, several works such as [15, 16, 17] are done. Information of current database state, especially, fill factor distribution of each table and index in real time may be useful for the fill factor control method we considered.

Recent approaches about physical design of database [6, 18, 5] aim to autonomic database systems analyzing workload pattern and deciding physical design parameters suitably. These methods are tend to be used for static physical structure, and are complementary with our strategies which is used for dynamic structural mutation.

6 Conclusion

This paper analyzed the impact of structural deterioration on OLTP performance degradation through the several experiments. The impact is that decrease of fill factor in a table can enlarge IOPS of accessing the records in the table and response time of involved transactions. Using this knowledge, we considered the page split strategies in order to keep high fill factor in structural mutation of B+tree implemented as a table/index of database, being adaptive to database access pattern.

In future work, a suitable model of database access pattern and a method extracting it online should be constructed, then the considered ideas above should be evaluated.

Acknowledgment

This research is supported by Leading Project 'e-society' under Ministry of Education, Culture, Sports, Science and Technology in Japan (MEXT). Advance Storage is a subproject of e-society, joint research between University of Tokyo and Hitachi Ltd.

References

[1] T. Johnson and D. Shasha. B-Trees with Inserts and Deletes: Why Free-at-Empty Is Better Than Merge-at-Half. *Journal of Computer and System Sciences*, 47(1):45–76, 1993.

[2] D. B. Lomet. Simple, Robust and Highly Concurrent B-trees with Node Deletion. In *ICDE*, pages 18–28, 2004.

[3] R. A. Hankins and J. M. Patel. Effect of node size on the performance of cache-conscious B$^+$-trees. In *SIGMETRICS*, pages 283–294, 2003.

[4] D. DeWitt and J. Gray. Parallel database systems: the future of high performance database systems. *Communications of the ACM*, 35(6):85–98, 1992.

[5] S. Agrawal, V. R. Narasayya, and B. Yang. Integrating Vertical and Horizontal Partitioning Into Automated Physical Database Design. In *SIGMOD Conference*, pages 359–370, 2004.

[6] D. C. Zilio, J. Rao, S. Lightstone, G. M. Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden. Db2 design advisor: Integrated automatic physical database design. In *VLDB*, pages 1087–1097, 2004.

[7] (Ed.)D.Lomet. Special Issue on Online Reorganization. *IEEE Data Eng. Bull.*, 19(2):1, 1996.

[8] C. Zou and B. Salzberg. On-Line Reorganization of Sparsely-Populated B+-Trees. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pages 115–124, 1996.

[9] TPC: Transaction Processing Performance Council. TPC BENCHMARK™ C Standard Specification. http://www.tpc.org/.

[10] T. Hoshino, K. Goda, and M. Kitsuregawa. A Study on Performance Analysis of Structural Deterioration for RDBMS. In *SMDB Workshop*, pages 81–90, 2005.

[11] MySQL: The World's Most Popular Open Source Database. http://www.mysql.com/.

[12] Seagate Technology LLC. *Cheetah 18LP FC Disk Drive Product Manual Volume 1*, 1999.

[13] S. Watanabe and T. Miura. Reordering B-tree files. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 681–686. ACM Press, 2002.

[14] J. S. Park and V. Sridhar. Probabilistic Model and Optimal Reorganization of B+-Tree with Physical Clustering. *IEEE Transactions on Knowledge and Data Engineering*, 9(5):826–832, 1997.

[15] S. Chaudhuri, G. Das, and U. Srivastava. Effective Use of Block-Level Sampling in Statistics Estimation. In *SIGMOD Conference*, pages 287–298, 2004.

[16] A. Aboulnaga, P. J. Haas, S. Lightstone, G. M. Lohman, V. Markl, I. Popivanov, and V. Raman. Automated Statistics Collection in DB2 UDB. In *VLDB*, pages 1146–1157, 2004.

[17] D. Narayanan, E. Thereska, and A. Ailamaki. Continuous resource monitoring for self-predicting dbms. In *MASCOTS*, pages 239–248, 2005.

[18] D. C. Zilio, S. Lightstone, and G. M. Lohman. Trends in automating physical database design. In *INDIN*, pages 441–445, 2003.