Storage Fusion

Masaru Kitsuregawa
Institute of Industrial Science
The University of Tokyo
kitsure@tkl.iis.u-tokyo.ac.jp

Kazuo Goda
Institute of Industrial Science
The University of Tokyo
kgoda@tkl.iis.u-tokyo.ac.jp

Takashi Hoshino
Graduate School of Information Science and Technology
The University of Tokyo
hoshino@tkl.iis.u-tokyo.ac.jp

ABSTRACT

So far, the core component of the IT system was absolutely a server, and the storage was recognized as its peripheral. The recent evolution of device and network technologies has enabled storage consolidation, by which all the data and its related simple software codes can be placed in one place. Storage centric designs are being deployed into many enterprise systems. The role of the storage should be reconsidered. This paper presents activities of *the Storage Fusion Project*, a five-year research and development project. *Storage Fusion* is an idea of elegant deep collaboration between storage and database servers. Two substantial works are presented in this paper. First, the exploitation of query execution plans enables dynamically informed prefetching, accordingly boosting ad-hoc queries significantly. Second, the idea of putting autonomic database reorganization into the storage has the potential benefit of relieving the management burdens of database structural deterioration.

1. INTRODUCTION

Although the magnetic disk drive has increased its areal density by eight orders of magnitude [13] during the fifty-year history [33], its basic operations have not almost changed. The disk drive, or a storage system which comprises multiple disk drives, is still a passive peripheral device which mainly stores data based on block read/write commands issued by the server. However, technological innovations that have happened in storage technologies for the past decade are giving us opportunities to reconsider the role of the storage.

First, the evolution of device technologies has enabled the vendors to accommodate plenty of computational resources into their storage products. A recent high-end disk storage [11, 35], which connects hundreds of disk drives to a hundred processors and tens of GB memory, is often deployed into enterprise systems. The computational capability of the system is significantly larger than that of disk drives and might be comparable with that of the server. But, all of such system resources are not necessarily fully busy. Utilizing idle portion of them for more sophisticated operations other than simple block reads/writes is a reasonable approach.

Second, Fibre Channel [12] released in the late 1990s drastically changed the physical connection between the storage system and the server. So far, a storage device was subsidiarily connected to a server. Its storage space was absolutely managed by the server and the stored data was also managed by each application separately. In contrast, storage networks have provided the storage device with the capability of connecting with other storage devices and/or servers. In the networked storage environment, consolidating the related data in one place looks straightforward. Encapsulating storage device management into the storage system could relieve the administration burden of each server.

Third, storage virtualization [6], which creates virtual storage space from physical storage devices, is a key software function. The consolidated storage space is shared by many servers. Managing its resources at separate server level is not practical. Moving resource management from the server to virtualization facilities implemented in the storage system could help easy optimization of the resource management.

The system design is shifting from a server-centric manner into a storage-centric manner; a gigantic high-end storage is connected via high-speed storage area networks and its storage space is exported by virtualization facilities. Yet, the storage system is still working passively for the server system; the system is just reacting block read/write commands. Recent storage systems have sufficient resources for executing sophisticated software codes. Colocating the data and its related software codes in a consolidated storage system may be a natural idea [2, 7, 10].

The Storage Fusion Project is a five-year research and development project we started in 2003. *Storage Fusion* is an approach of deep collaboration between storage and database servers. If we elegantly cut some portion of the software functions from the server and put it to the storage system, the storage system could work much more closely with the server. Of course, new interfaces might become necessary, but they could be justified by the benefits. The main objective of the project is to explore such elegant fusion and to clarify the potential benefits.

This paper concisely presents two substantial works of the projects. First, we discuss the exploitation of query execution plans for boosting disk storage. Second, we present the approach of putting database reorganization into a storage system. Brief experimental evaluations follow each study in the paper.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

The remainder of the paper is organized as follows. Section 2 discusses query execution plan assisted advanced prefetching and Section 3 presents self-reorganizing storage system. Section 4 summarizes related works and Section 5 concludes the paper.

2. DATABASE SERVER ASSISTED STORAGE BOOSTING

2.1 Exploitation of Query Execution Plans

Researchers at University of California, Berkeley reported, in 2000, that the amount of digital data newly generated by human beings had been doubled every year [37]. Last year, IDC's analysts forecasted that 161EB of data has been created in 2006 and 988EB would be so in 2009 [17]. U.C. Berkeley's estimate looks almost supported by this new report. Such explosive information expanding is often seen in many places. For example, sensor network technology has enabled precise monitoring of real world events. Significantly large amount of monitored data is continuously being stored into storage systems. Deep investigation of such massive data may open a gate for real world analysis. Some applications are already deployed. Retail business companies such as Wal-Mart started putting IC tags on their sales items [3, 39]. Efficient distribution of sales goods is a key to their business success. Analyzing all the business processes such as manufacturing, logistics and sales should be helpful, or might be even essential, to business administration.

Performance of data intensive applications such as deep analysis on data warehouses is strictly limited by the aggregate capability of storage systems in which the target data is stored. Unfortunately, disk drives have gained 7% performance improvement for random accesses every year [13]. The digital data is exponentially expanding, but its processing throughput is only slightly increasing. Bridging the gap is a big research issue.

In the Storage Fusion Project, we explored the exploitation of high-level behavioral information of software programs which are running on the server. When a database server receives a query, it generates a query execution plan and then processes the query on the basis of the generated plan. If the storage system is informed of the plan information, it can much more precisely predict the data which will be accessed by the server in near future. Query execution plan assisted advanced prefetching is an idea of issuing prefetching commands based on such informed knowledge.

2.2 Query Execution Plan Assisted Advanced Prefetching

Figure 1 illustrates an overview of query execution plan assisted advanced prefetching. When a user program issues a query, a database server generates a query execution plan. The generated plan is immediately sent to an advanced prefetching facility, which is implemented in a storage subsystem. Then, the database server begins to process the query based on the plan. While the query is being processed, the advanced prefetching facility is informed of address information of each read request issued by the server. By analyzing the plan and the address information, the prefetching facility can predict the data which is to be accessed by the server. Prefetching such data would significantly improve the query processing performance.

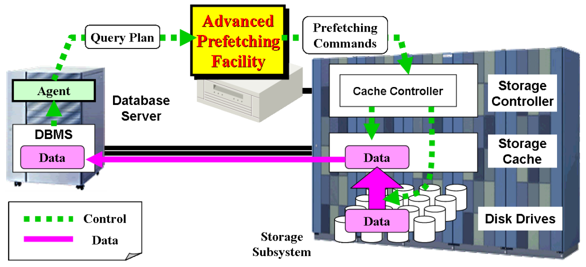


Figure 1: An Overview of Query Execution Plan Assisted Advanced Prefetching.

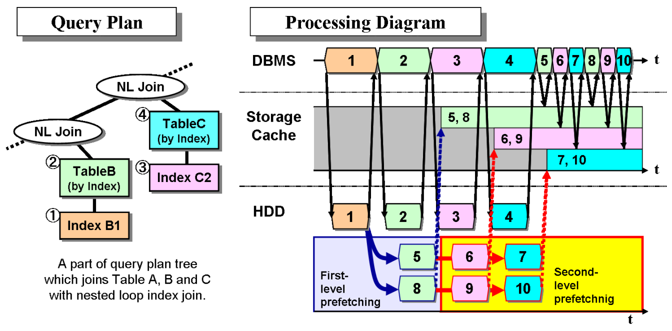


Figure 2: First-level Prefetching and Second-level Prefetching.

Prediction and prefetching is done in the two steps, which are also illustrated in Figure 2.

First-level Prefetching. Based on the informed address, the advanced prefetching facility identifies index records (Index B1) which will be accessed, and then tries to read the index records in a prefetching manner. When the index records are read to the cache, the facility analyzes the records to identify table records (Table B) to which the index records are associated. Finally, the facility issues prefetching commands to the table records again.

Second-level Prefetching. The advanced prefetching facility analyzes the table records prefetched by first-level prefetching for enabling further prefetching. That is, the facility tries to identify index/table records (Index C2 and Table B) which will be joined with the already prefetched table, and then issues prefetching commands to the new index/table records. Multiple processes of second-level prefetching can be executed; another prefetching can be triggered based on the data prefetched by previous prefetching.

2.3 Coordinating Multiple Queries

The proposed prefetching is an approach to control IO requests at storage level using informed knowledge. If only a single query is processed in the system, this would work well. However, usually a IO command conveyed on a Fibre Channel network is not aware of its semantics. That is, the IO command includes only a low-level initiator identifier such as Fibre Channel name instead of higher-level identifiers. When the storage subsystem receives a IO request, it cannot identify an application, a query, a transaction, a process or a thread which has issued the IO request. If multiple queries were being processed in the system, the advanced prefetching facility could not identify its source query. Our

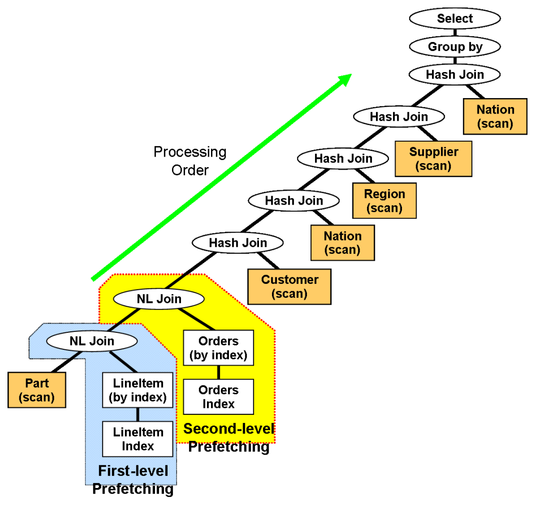


Figure 3: A Query Execution Plan of Query 8.

solution is to put a query identifier on each read command. By analyzing the identifier, the advanced prefetching facility can identify a query for each IO request, so that it can trigger prefetching appropriately for a separate query.

2.4 Experimental Evaluations

We implemented prefetchning function using Hitachi's storage subsystem and developed query execution plan assisted advanced prefetching for a commercial database system, HiRDB [14].

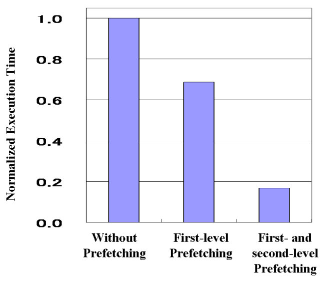
For evaluations, we prepared TPC-H dataset (SF=3.0) and measured execution time of query 8 on different configurations. The generated query execution plan is depicted in Figure 3.

Figure 4(a) presents the result, which compares three cases: conventional query processing, query processing with only first-level prefetching enabled, and query processing with first- and second-level prefetching enabled. We could gain six times faster query processing totally by the advanced prefetching. The performance improvement was investigated further. Figure 4(b) presents microscopic IO behaviors that were traced during the first one-second execution in each. A red point and a blue point denote respectively a prefetching read command and a normal read command in the graphs. IO requests were sparsely issued in the conventional query processing, but the proposed prefetching technique could concentrate the IOs. We measured cache hit ratios of the storage subsystem. Only 20% of reads could hit in the conventional query processing, but the hit ratio could improve to 90% by the advanced prefetching.

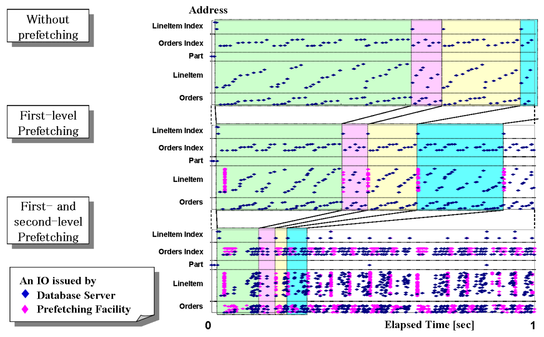
We also evaluated the advanced prefetching for a multiple query processing environment. Figure 5 presents the result. Without the query identification tag, the performance improvement degraded for two or more concurrent queries. In contrast, with the query identification tag, such deterioration was not not observed.

2.5 Summary

The idea of exploiting query execution plans enables dynamically informed prefetching. We developed the advanced prefetching mechanism in the commercial storage subsystem and conducted validation experiments. Six-fold performance improvement was confirmed in a ad-hoc query processing.



(a) Execution Time



(b) IO Behavior

Figure 4: Results of Experiments. (Single Query)

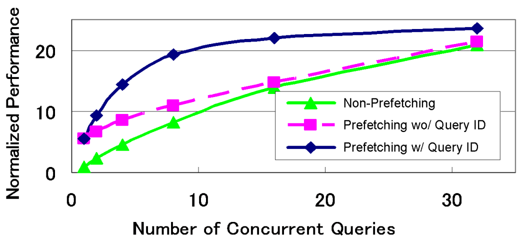


Figure 5: A Result of Experiments. (Multiple Queries)

3. SELF-REORGANIZING STORAGE SYSTEM

3.1 Database Reorganization

The data structure may gradually deteriorate as a number of records are inserted, deleted and/or updated. This phenomenon is called *structural deterioration*, which will degrade data access performance. Assume a B+-tree structure, where leaf pages are expected to be physically placed in key order. Given that many records are inserted into a particular page, the page splits and some records of the page are moved to another physically distant page. As such a page split occurs many times, correlation between record key and physical position gradually degrades, and thus finally a leaf page scan involves many disk seeks [38]. Such issues are serious in large-scale database. Database reorganization [21], which moves recodes in the storage space to remove structural deterioration, is a unique solution. Reorganization is recognized as an essential function for database systems. In fact, all of the major database systems have reorganization tools in their software suites [5, 15, 25, 27, 32].

Database administrators are responsible of managing performance of the database by executing database reorganization only when structural deterioration degrades the database performance [20, 23, 30, 40]. However, such careful management of structural deterioration does not look feasible in practice. Rather, database reorganization is a typical headache department for database administrators. The difficulties of structural deterioration management can be grouped into two issues.

Data Intensiveness. So far the administrator used to be allowed to execute database reorganization in a offline way, so that database reorganization could not interfere with the online workloads. But, service availability is a prime concern today. Most of database servers need to be servicing for 24 hours. The database must be reorganized in a online fashion. However, database reorganization is very time consuming, issuing a massive number of disk accesses. In addition, we should consider recent IT systems, in which rich bandwidth is prepared for internal communication within each subsystem but inter-subsystem bandwidth is rather limited. If we run database reorganization online on the database server, the storage network connection between the storage and the server would be congested. The user would see much lower transaction processing performance.

Deterioration Measurement. Database administrators are responsible of carefully scheduling database reorganization in order to manage the system performance; specifically, the administrator should determine the portion of the database to be reorganized and the time point to trigger reorganization. However, database systems and recent storage virtualization facilities abstract data storage structures. This looks beneficial to users; they do not have to consider physical data storage and they can manage data by the use of higher-level description such as SQLs. However, the database is also a black box to the administrator at the same time. Structural deterioration is not clearly visible in many cases although the information of structural deterioration is crucial to optimize database reorganization. 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 high-end systems may opt to make reorganization plans based on rules of thumb and rough performance statistics. Such naive solutions lead to inefficient and expensive database administration.

In this paper, we discuss an idea of executing online database reorganization in the storage system. Assuming that database reorganization could be implemented in the storage system, the database server would no longer have to consider reorganization. This must be good news for the administrator. In addition, database reorganization moves huge amount of data. Server-side implementation would suffer from non-negligible overheads due to multiple virtualization facilities. Storage-level implementation is a promising approach.

The paper presents the design overview and the prototype implementation of a self-reorganizing storage system (SRS), a highly functional disk storage which has the capability of autonomic database reorganization. To clarify the potential benefit of our approach, the experimental results are also briefly presented.

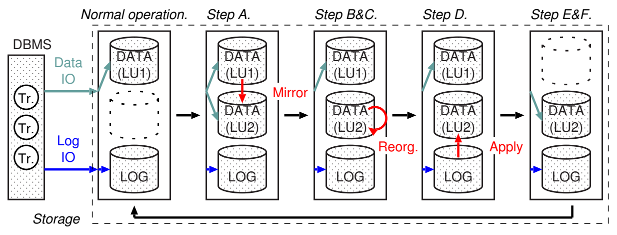


Figure 6: Storage-level Split Database Reorganization.

3.2 Storage-level Split Database Reorganization

The self-reorganizing storage system has the capability of reorganizing database which is stored in the system itself. The reorganization is performed in the the following procedure as illustrated in Figure 6. (A.) First, the storage system dynamically allocates a new tablespace, which mirrors an original tablespace[1]. (B.) Next, the database server quiesces the tablespace temporarily, and the storage system splits the mirrored tablespaces. Then, the database server mounts the original tablespace and starts transaction processing again. (C.) The storage system reorganizes the copied tablespace, and subsequently (D.) applies the database log to the reorganized tablespace so that the reorganized tablespace can logically catch up with the original tablespace. (E.) The database server quiesces the tablespace temporarily again, mounts the reorganized tablespace, and then restarts transaction processing. (F.) Finally, the storage system discards and releases the original tablespace.

In the above steps, (C.) reorganization and (D.) log catch-up, which are new functions for storage systems, are very data intensive. Focusing on accelerating these steps, we have introduced two techniques: physical address level data movement and eager log compaction.

Storage space of disk drives is exported to server applications through multiple virtualization layers such as LVMs, virtualization switches and disk array controllers, each of which does logical-physical address translation. The data path from the server application to the disk drive is very long, and when the application tries to move large data, accumulated overheads of these virtualization facilities are not negligible. We try to improve the processing efficiency by introducing physical-address-level data movement; the reorganizer can process database pages at physical address level. In addition, the virtualization facility is usually implemented with finite-length queues, in which IO commands and transferred data are managed. Each virtualization layer manages such finite resources within itself, not considering other layers. Thus, behavior of the whole system is far from global optimization. In many cases, *impedance mismatch* occurs in such multiple-level queuing systems. The reorganization facility can directly schedule IOs at physical address level. For example, fine-gained request elevation and consecutive block access coalescence greatly help throughput improvement.

Eager log compaction is a key technology of performance improvement of the log applier. Each entry of database log has a unique LSN (log sequence number) and normal

[1]To simplify the description, we assume that an LU (logical unit) is directly used for a tablespace.

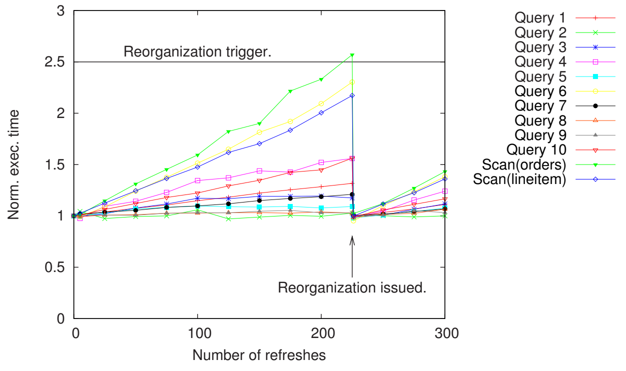


Figure 7: A Case Study of Database Reorganization.

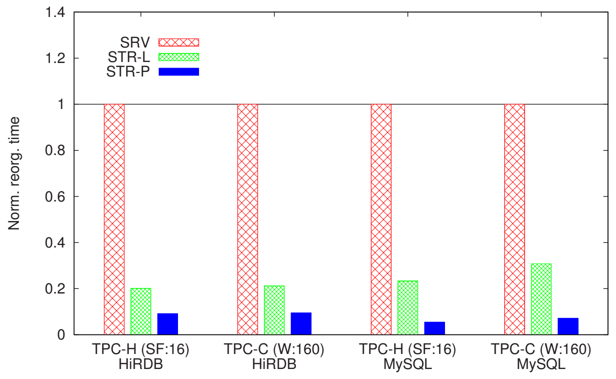


Figure 8: Comparison of Reorganization Time.

redo operations apply the log entries in LSN order. In our method, the log applier compacts the log sequence in a *log window buffer*, and then applies the compacted sequence to the tablespace. The compaction process comprises *log folding* and *log sorting*. Log folding is a technique to reduce the number of log entries to be applied. The log entries which manipulate the identical data are coalesced in the log window. On the other hand, log sorting reorders log entries in the log window to improve disk access sequentiality. In many cases, an entry of database log has a physical reference to the target record. Such physical address information could be helpful for scheduling [22,36]. Both the methods together can reduce the log catch-up cost, boosting log catch-up.

3.3 Prototype Implementation and Evaluations

We implemented a prototype of the self-reorganizing storage system. The prototype was developed using two different database systems: HiRDB [15], a commercial product, and MySQL (with InnoDB storage engine) [26], an open-source product. The reorganization software is composed of multiple threads. In order to reduce processing overheads and improve disk access parallelism, our implementation deploys parallel pipelined data processing. For example, unloading, external sorting[2] and loading are processed for database reorganization in a pipelined manner as much as possible.

Here we present a brief case study of database reorganization. Using TPC-H on HiRDB, we investigated the effect of structural deterioration on query performance and the performance restoration by database reorganization. Figure 7 shows gradual performance degradation of typical ad-hoc queries. This degradation was due to uncollected garbage space which was produced by the refresh functions. The graph shows that reorganization was initiated based on the specified threshold and then the performance could improve to the original level.

We also evaluated the database reorganization time for different datasets, TPC-H and TPC-C, and different database systems, HiRDB and MySQL. For comparison, we measured three cases[3]: **SRV**, **STR-L** and **STR-P**. **SRV** denotes conventional reorganization tools. We used `pdrorg` for HiRDB and `mysqldump / mysqlimport` for MySQL. In contrast, **STR-L** and **STR-P** denote the self-reorganizing storage system. **STR-P** enabled physical address awareness whereas **STR-L** did not so.

[2]Sort run length depends on available memory size.

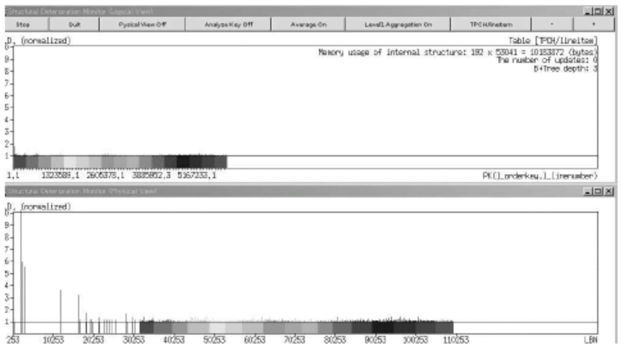
[3]Note that we prepared the same IO bandwidth for each configurations for fairness, although the external bandwidth of real storage systems is usually much lower than the internal.

Figure 8 summarizes the results, which show that the storage-level database reorganizer significantly overperformed. First, we analyzed **SRV** and **STR-L**. Compared with the conventional server-side reorganizers, the storage-level reorganizer could improve the performance by 80% for HiRDB and 74% for MySQL. One possible reason is that little effort has been exerted for performance improvement of conventional reorganization software, because the external IO bandwidth may be congested by such server-side solutions. In our storage-level split strategy, large data transfer is performed within the storage system, and intensive data movement is enabled to leverage the internal bandwidth. We believe that the proposed parallel data processing technology contributed to this advantage. Next, we compared **STR-L** and **STR-P** to investigate the effect of physical address awareness. Compared with the reorganization which does not consider physical address, the physical address level data movement could improve the performance by 55% for HiRDB and 76% for MySQL. Interestingly, although noticeable performance improvement was gained even in **STR-L** case, further improvement was achieved by physical address awareness in **STR-P** case. In total, by delegating database reorganization into the storage system and introducing parallel data processing and physical address awareness, we achieved significant performance improvement, 91% for HiRDB and 94% for MySQL.

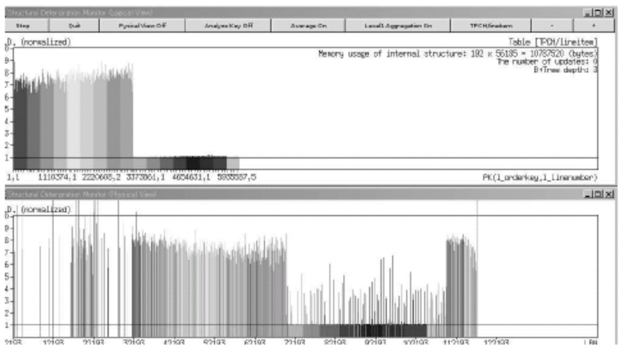
3.4 Online Monitoring of Database Structural Deterioration

In the case study described in the previous section, we triggered database reorganization based on the threshold of execution time. This strategy looks applicable to many cases, but more direct monitoring of database structural deterioration would be helpful and might be necessary for several cases. Here we present an online monitoring facility we explored for enabling such monitoring [16].

The monitor is composed of a *sniffer* and an *estimator*. The sniffer, implemented in low-level storage engine of database systems, captures specific events such as page split and sends the event to the estimator. Upon the received information, the estimator calculates the degree of the structural deteri-



(a) after data load



(b) after 50 refreshes

Figure 9: Online Monitor of Database Structural Deterioration.

oration for a considered access on the basis of a given structure model. Here the degree of structural deterioration is defined as a deterioration ratio of predicted disk access time. Let us suppose a range scan of a relational table. If the scan takes 300 seconds in the well-organized database and 450 seconds in the deteriorated database, then the deterioration degree becomes 1.5.

The monitoring facility has three technical advantages. First, the event can be captured for each concerned page. If only a limited section of the database is deteriorated, the deteriorated section can be identified almost at page level. Second, physical performance characteristics such as seek profiles of disk drives are considered in calculating the deterioration degree. Thus, highly accurate estimation of the structural deterioration is expected. Finally, the sniffer only needs to capture a small number of events and the estimator can update the measured structural deterioration in an incremental fashion, instead of performing full table scan. Monitoring overheads can be minimized.

We developed a monitoring software suite for MySQL on Linux operating system. The suite includes a graphical tool which has the capability of visualizing structural deterioration based on the monitored information. Figure 9 shows an example of structural deterioration which was observed in TPC-H benchmark.

3.5 Summary

We show a design and a prototype implementation of a self-reorganizing storage system. Physical address awareness and eager log compaction are key technology to fully utilize the rich IO processing power of recent high-end disk storage. The presented case study and the performance experiments validate the potential benefits. In addition, we present a online monitoring facility of database structural deterioration.

4. RELATED WORKS

The idea of placing high-level software codes onto storage processors can be traced back to the database machines of the 1970s and 1980s [9, 28, 31, 34]. Because expensive dedicated hardware could not be justified by its performance gains, the vendors finally withdrew from database machine development. In the late 1990s, Active Disks and Intelligent Disk were proposed [1, 19, 29]. Those ideas, which per se were similar to the database machines, were supported by the technology background of increasing performance and decreasing cost of processors and memory. Those researches tried to run ad-hoc queries, data mining, etc. on storage processors. Delegating such core database codes into the storage is one approach, but it is still not realistic at present.

The idea of the Storage Fusion Project differs from these past works. We still opt to execute queries and transactions on server processors. Rather, our objective is to derive elegant deep collaboration between storage and database servers. Query execution plan assisted advanced prefetching should be seen as an attempt to move only IO optimization from the server to the storage. The storage resources are virtualized and shared by many servers. Optimizing IOs at separate server looks far from global optimization. Arranging IOs at storage level using more rich information would be much more efficient. Self-reorganizing storage is an approach of moving database reorganization into the storage system. Compared with query or transaction processing, such reorganization codes can be more easily cut off database systems and not be too complicated to be executed on the storage processors. In addition, database reorganization is much data intensive. Executing the software code closely to disk drives are advantageous than server-level implementation. A new type of reasonable function partitioning can be confirmed.

New interfaces look necessary in order to deploy our solution into commercial products. We should consider this issue more carefully. Vendors and researchers have been studied new high-level protocols such as SMI-S [8] and OSD [24], which should support our approach. In fact, simple functions are being implemented into commercial storage products and such solutions are widely accepted [2, 7, 10, 18]. In addition, we are now considering the possibility of using recent virtual machine technology, where the software execution environment can be isolated and can flexibly migrate between various machines. If we deploy such operating system capabilities such as LPAR [4] in the processors of disk storage systems, our approach may become straightforward.

5. CONCLUSION

This paper presents research activities of *the Storage Fusion Project*, in which we have studied elegant deep collaboration between storage and database servers. Two substantial works are presented in this paper. Query plan assisted advanced prefetching can significantly improve the performance of ad-hoc queries. We confirmed six-fold speedup in the experiment. The approach of self-reorganizing storage can encapsulate database reorganization into storage systems and can boost database reorganization by approximately ten times.

6. ACKNOWLEDGMENTS

This work has been supported in part by the leading project *e-Society* under the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan. The Storage Fusion Project (formally *Advanced Storage*), a subproject of e-Society, is a joint research between The University of Tokyo and Hitachi Ltd. The project has been contributed by Mamoru Sugie, Toshihiko Odaka, Soichi Oyama, Hiroyoshi Tsuchiya, Nobuo Kawamura, Kazuhiko Mogi, Shinji Fujiwara, Yoshio Suzuki, Satoru Watanabe, Kazuhiko Mizuno, Yuichi Yagawa, Norifumi Nishikawa, Hideomi Idei, Naokazu Nemoto, Kota Yamaguchi, Takao Satoh, Jun Abe and Takashi Oeda. Takashi Hoshino is partially supported by Research Fellowships of the Japan Society for the Promotion of Science (JSPS) for Young Scientists.

7. REFERENCES

[1] A. Acharya, M. Uysal, and J. Saltz. Active Disks: Programming Model, Algorithm and Evaluation. In *Proc. Int'l. Conf. on Arch. Suport for Prog. Lang. and Operating Syst.*, pages 81–91, 1998.

[2] A. Azagury, M. E. Factor, J. Satran, and W. Micka. Point-in-Time Copy: Yesterday, Today and Tomorrow. In *Proc. NASA/IEEE Conf. on Mass Storage Syst. Tech.*, pages 259–270, 2002.

[3] C. Babcock. Data, Data, Everywhere. *InfomationWeek*, 2006.

[4] D. Barrick et al. Logical partitions on the IBM PowerPC, A Guide to Working with LPAR on Power5 i5 Severs. *IBM Redbooks*, 2005.

[5] BMC Software, Inc. Easing the Pain of an IMS Reorganization. White Paper, 2002.

[6] F. Bunn and R. Peglar. Storage Virtualization I. What, Why, Where and How? SNIA Education, 2004.

[7] J. Carleton. Electronic Data Archiving and Retrieval in the Public Sector. Analyst Report, Frost & Sullivan, 2004.

[8] M. A. Carlson and J. Crandall. Storage Management from SMI-S to Management Frameworks. SNIA Education, 2007.

[9] D. J. DeWitt and P. B. Hawthorn. A Performance Evaluation of Database Machine Architectures. In *Proc. Int'l. Conf. on Very Large Data Base*, pages 199–214, 1981.

[10] EMC Corp. EMC Mainframe Solutions Guide. Engineering White Paper, 2002.

[11] EMC Corp. EMC Symmetrix DMX Series. White Paper, 2004.

[12] Fibre Channel Industry Association. FCIA Roadmap. http://www.fibrechannel.org/.

[13] Hitachi Global Storage Technologies, San Jose Research Center. HDD Technology Overview Charts, 2003.

[14] Hitachi Ltd. Hitachi HiRDB Version 7. `http://www.hitachi.co.jp/Prod/comp/soft1/hirdb/`.

[15] Hitachi Ltd. Hitachi Relational Database Management System Solutions for Disaster Recovery to Support Business Continuity. Review Special Issue, Hitachi Technology, 2004.

[16] T. Hoshino, K. Goda, and M. Kitsuregawa. Online Monitoring of Database Structural Deterioration. In *Proc. Int'l Conf. on Autonomic Comput.*, pages 22–23, 2007.

[17] IDC. The Expanding Digital Universe: A Forecast of Worldwide Information Growth Through 2010. An IDC White Paper sponsored by EMC, 2007.

[18] M. Ji, A. Veitch, and J. Wikes. Seneca: remote mirroring done write. In *Proc. USENIX Conf. on File and Storage Tech.*, pages 253–268, 2003.

[19] K. Keeton, D. A. Patteson, and J. M. Hellerstein. A case for intelligent disks (IDISKs). *SIGMOD Record*, 27(3):42–52, 1998.

[20] G. M. Lohman and J. A. Muckstadt. Optimal Policy for Batch Operations: Backup, Checkpointing, Reorganization, and Updating. *ACM Trans. Database Syst.*, 2(3):209–222, 1977.

[21] D. Lomet, editor. *IEEE Data Eng. Bull.: Special Issue on Online Reorganization.*, volume 19. IEEE Computer Society, 1996.

[22] C. R. Lumb, J. Schindler, G. R. Ganger, D. F. Nagle, and E. Riedel. Towards higher disk head utilization: extracting free bandwidth from busy drives. In *Proc. USENIX Symp. on Operating Syst. Design and Imple.*, pages 87–102, 2000.

[23] K. Maruyama and S. E. Smith. Optimal Reorganization of Distributed Space Disk Files. *Comm. ACM*, 19(11):634–642, 1976.

[24] M. Mesnier, G. R. Ganger, and E. Riedel. Object-based Storage. *IEEE Comm.*, 41(8):84–90, 2003.

[25] A. Mohamed, G. Candia, and D. Sherwin. Comparing Architectures of Online Reorganization. White Paper, Quest Software, 2002.

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

[27] Oracle Corp. Oracle Database 10g Online Data Reorganization & Redefinition. White Paper, 2004.

[28] E. A. Ozkarahan, S. A. Schuster, and K. C. Smith. RAP - Associative Processor for Database Management. In *Proc. AFIPS Conf.*, pages 379–387, 1975.

[29] E. Riedel, G. A. Gibson, and C. Faloutsos. Active Storage For Large-Scale Data Mining and Multimedia. In *Proc. Int'l. Conf. on Very Large Data Base*, pages 62–73, 1998.

[30] B. Shneiderman. Optimum Data Base Reorganization Points. *Comm. ACM*, 16(6):362–365, 1973.

[31] D. L. Slotnick. Logic per Track Devices. *Advances in Computers*, 10:291–296, 1970.

[32] G. H. Sockut, T. A. Beavin, and C.-C. Chang. A Method for On-Line Reorganization of a Database. *IBM Syst. J.*, 36(3):411–436, 1997.

[33] L. D. Stevens. The Evolution of Magnetic Storage. *IBM Res. Develop.*, 25(5):663–676, 1981.

[34] S. Y. W. Su and G. J. Lipovski. CASSM: a cellular system for very large database. In *Proc. Int'l. Conf.*

on Very Large Data Base, pages 456–472, 1975.

[35] N. Takahashi and H. Yoshida. Hitachi TagmaStore Universal Storage Platform: Virtualization without Limits. White Paper, Hitachi Ltd., 2004.

[36] E. Thereska, J. Schindler, J. S. Bucky, B. Salmon, C. R. Lumb, and G. R. Ganger. A framework for building unobtrusive disk maintenance applications. In *Proc. USENIX Conf. on File and Storage Tech.*, pages 213–226, 2004.

[37] University of California. How Much Information?, 2000.

[38] S. Watanabe and T. Miura. Reordering B-tree Files. In *Proc. ACM Symp. on Applied Comput.*, pages 681–686, 2002.

[39] R. Whiting. Data Avalance. *InfomationWeek*, 2004.

[40] S. B. Yao, K. S. Das, and T. J. Teorey. A Dynamic Database Reorganization Algorithm. *ACM Trans. Database Syst.*, 1(2):159–174, 1976.