論文 / 著書情報
Article / Book Information

Title	A Concurrency Control Method for Parallel Btree Structures
Author	Tomohiro Yoshihara, Dai Kobayashi, Ryo Taguchi, Haruo Yokota
Journal/Book name	Proc. of International Special Workshop on Databases For Next Generation Researchers (SWOD 2006), Vol. , No. , pp. 71-76
Issue date	2006, 4
DOI	10.1109/ICDEW.2006.7
URL	http://www.ieee.org/index.html
Copyright	(c)2006 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other users, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works for resale or redistribution to servers or lists, or reuse of any copyrighted components of this work in other works.
Note	このファイルは著者（最終）版です。 This file is author (final) version.

A Concurrency Control Method for Parallel Btree Structures

Tomohiro Yoshihara[1] Dai Kobayashi[1] Ryo Taguchi[2] Haruo Yokota[3,1]

[1]Department of Computer Science, Graduate School of Information Science and Engineering,
Tokyo Institute of Technology 2–12–1 Ookayama, Meguro-ku, Tokyo 152–8552, Japan

[2]Science and Technical Research Laboratories, Japan Broadcasting Corporation
1–10–11 Kinuta, Setagaya-ku, Tokyo 157–8510, Japan

[3]Global Scientific Information and Computing Center, Tokyo Institute of Technology
2–12–1 Ookayama, Meguro-ku, Tokyo 152–8550, Japan

{yoshihara,daik}@de.cs.titech.ac.jp, taguchi.r-cs@nhk.or.jp, yokota@cs.titech.ac.jp

Abstract

A new concurrency control protocol for parallel Btree structures, MARK-OPT, is proposed. MARK-OPT marks the lowest structure-modification-operation (SMO) occurrence point during optimistic latch coupling operations, to reduce the cost of SMO compared to the conventional protocols such as ARIES/IM and INC-OPT. The marking reduces the frequency of restarts for spreading the range of X latches, which will clearly improves the system throughput. Moreover, the MARK-OPT is deadlock free and satisfies the physical consistency requirement for Btrees. These indicate that the MARK-OPT is right and suitable as a concurrency control protocol for Btree structures. This paper also proposes three variations of the protocol, INC-MARK-OPT, 2P-INT-MARK-OPT and 2P-REP-MARK-OPT, by focusing on tree structure changes from other transactions. We implement the proposed protocols, the INC-OPT, and the ARIES/IM for the Fat-Btree, a form of parallel Btree, and compare the performance of these protocols using a large-scale blade system. The experimental results indicate that the proposed protocols always improve the system throughput, and the 2P-REP-MARK-OPT is the most useful protocol in a high update environment. Moreover, in the experiment, the low frequency of restarts in the proposed protocols indicates that the marking in the proposed protocols is effective.

1 Introduction

In a shared-nothing parallel machine for database systems, retrievals and updates are performed in parallel on a processing element (PE) storing the object data. Bottlenecks on highly accessed PEs by access-request skew degrade system performance. To improve the performance, data partitioning methods are significant [3, 4]. The value range partitioning method with a parallel Btree structure is an excellent approach to handle the skews, as it also provides clustering I/Os and fast access paths for both exact match and range queries.

To make the parallel Btree practical, it is important to consider the cost of update operations requiring concurrent accesses to multiple PEs. If all PEs have copies of a single Btree, the synchronization between the PEs degrades the system throughput considerably. On the other hand, if all index nodes of a Btree are only placed on a single PE, the PE becomes a bottleneck in parallel processing due to the concentration of all index accesses to it. To resolve these problems, an update conscious parallel Btree structure, Fat-Btree, has been proposed, and experimental results indicate that Fat-Btrees provide better performance than other parallel Btrees [12].

It is also important to provide an efficient concurrency control protocol for parallel Btree structures, including Fat-Btree. The INC-OPT protocol suited for a parallel Btree on a shared-nothing parallel machine has been proposed [8]. The INC-OPT protocol outperforms the conventional Btree concurrency control protocols, such as the B-OPT protocol [1] and the ARIES/IM protocol [9]. However, the costs of spreading an X latch in the protocol are still high when structure modification operations (SMOs) occurred frequently, which degrades the total performance.

In this paper, we propose a new concurrency control protocol for parallel Btree structures, MARK-OPT. It reduces the frequency of restarts retraversing from the root node compared with the INC-OPT protocol. We also propose three variations of the protocol, INC-MARK-OPT, 2P-INT-MARK-OPT and 2P-REP-MARK-OPT, focusing on the tree structure changes caused by other transactions.

We implemented the four proposed protocols, the INC-

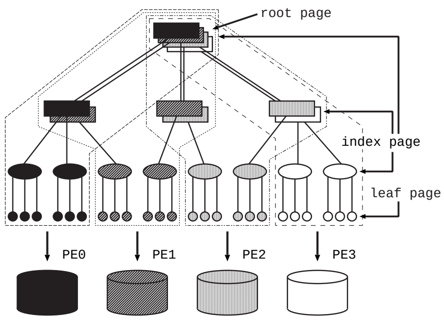

Figure 1. Fat-Btree

OPT and the ARIES/ IM[1] on an autonomous disk system [11] using the Fat-Btree as the distributed directory structure and measured the system throughput with changing update ratio. These experimental results indicate that the proposed protocols are effective.

The remainder of the paper is organized as follows. First, the concept of the Fat-Btree and the concurrency controls for parallel Btrees are introduced as background in section 2. We propose the new concurrency control protocols for parallel Btree structures in section 3. The experimental results are reported in section 4. The final section presents the conclusions of this paper.

2 Background

2.1 Fat-Btree Structure

The Fat-Btree [12] is a form of parallel Btree in which the leaf pages of the B^+-tree are distributed among PEs and each PE has a subtree of the whole Btree, which contains the root node and intermediate index nodes between the root node and leaf nodes allocated to the PE. Figure 1 shows an example of a Fat-Btree using four PEs.

Although the number of copies increases with proximity to the root node in a Fat-Btree, the update frequency of these nodes is relatively low. On the other hand, leaf nodes have a relatively high update frequency but have no copy. Consequently, the nodes to be updated more frequently have lower overhead for updating with respect to the synchronization between duplicated nodes.

Moreover, in the Fat-Btree, index pages are only necessary for searching for the leaf pages stored in each PE. Therefore, the Fat-Btree can have a high cache hit rate if the index pages are cached in each PE. Because of the high cache hit rate the update processes and the search processes can be processed quickly, compared with a conventional parallel Btree structure.

[1] We implemented ARIES/IM based on [9], but did not recovery mechanism of it because we do not focus on recovery in this paper.

Table 1. Latch matrix

Mode	IS	IX	S	SIX	X
IS	○	○	○	○	
IX	○	○			
S	○		○		
SIX	○				
X					

2.2 Concurrency Control Methods

Some concurrency control method for the Btree is necessary to guarantee its consistency. Instead of locks, fast and simple latches are usually used for concurrency control during traversing index nodes in a Btree [5]. A latch is a form of semaphore and the latch manager does not have a deadlock detection mechanism. Therefore, concurrency control for a Btree node should be deadlock free.

In this paper, a latch is assumed to have five modes: IS, IX, S, SIX, and X as shown in Table 1 [5]. The symbol of "○" means that the two modes are compatible.

Because a parallel Btree structure, including the Fat-Btree, has duplicated nodes a special protocol for the distributed latch manager is required to satisfy the latch semantics. Requested IS and IX mode latches can be processed only on a local PE, whereas the other modes have to be granted on all the PEs storing the duplicated nodes to be latches. That is, the IS and IX modes have much smaller synchronization cost than the S, SIX and X modes, which require communication between the PEs. The S, SIX, and X mode latches on remote copies are acquired by using their pointers. In addition, such latches have to be set in linear order to avoid dead lock. This means synchronization cost grows in proportion to the number of PEs related to latches.

The following conditions of the concurrency controls for parallel Btrees are proposed in [8].

Condition 1. A concurrency control method for parallel Btrees should satisfy the following conditions:

(a) No concurrency control protocol method for index nodes, which cause deadlocks, should be used.

(b) Use of S, SIX, and X mode latches on index nodes at upper levels of the Btree should be avoided as much as possible.

(c) The entire tree should not be latched, even for a short duration.

B-OPT [1], OPT-DLOCK [10], and ARIES/IM [9] are excellent concurrency control methods for a Btree on a single machine. However, they do not satisfy Condition 1: B-OPT does not satisfy Condition 1-(b), OPT-DLOCK does not satisfy Condition 1-(a), and ARIES/IM does not satisfy

Condition 1-(c). Therefore, these concurrency controls are unsuitable for parallel Btrees, such as Fat-Btree.

2.3 INC-OPT Protocol

The INC-OPT protocol satisfying Condition 1 has been proposed [8] as a concurrency control protocol for parallel Btrees, including Fat-Btree.

The INC-OPT protocol to search for a key is simple. An IS mode latch is held on the root node initially, then the following steps are performed during traversal in the parallel Btree:

1. Derive a pointer to a child node by comparing the key in the parent node.
2. Acquire an IS mode latch on the child, and release the latch on the parent.
3. Repeat the above steps until the traverse reaches a leaf node.

The above procedure is usually called latch-coupling [5]. When the traverse arrives at a leaf node, it acquires an S latch on the leaf and reads data from it.

The INC-OPT protocol for an update consists of two phases.

The first phase: The traverse reaches a leaf with latch-coupling with the IX latches. At the leaf, an X latch is acquired. If the leaf node is not full, the updater updates it. Otherwise, if the leaf node is full it splits, this latch is once released and it then shifts to the second phase.

The second phase: The INC-OPT tries to acquire the X mode latches on the lower two nodes, i.e., the leaf node and its parent, with the X mode latches. If the parent node also causes an SMO, it releases all latches and tries to acquire the X mode latches on the lower three nodes. This process continues until all the nodes involved by SMOs are protected by X latches.

The INC-OPT protocol is precisely defined in [8].

When an SMO occurs, the INC-OPT may need multiple restarts. When the root node causes an SMO, the INC-OPT requires as many phases as the height of the Btree. This increases the response time of the update operations. In addition, it decreases the system throughput because of the multiple X latches used on the index nodes at upper levels.

3 Proposed Protocols

3.1 MARK-OPT Protocol

We propose a new concurrency control protocol for parallel Btrees, including Fat-Btree, which marks the lowest SMO occurrence point during optimistic latch coupling operations. We call this the *marking optimistic (MARK-OPT) protocol*, which improves the response time by reducing the frequency of restarts. In addition, the MARK-OPT produces high system throughput because of the reduction of middle phases for spreading an X latch and removes needless X latches.

The procedure MARK-OPT uses to search for a key is identical to INC-OPT, whereas its update consists of the following two phases:

The first phase: The traverse reaches a leaf with latch-coupling with the IX latches. If the index node is full, MARK-OPT marks the height of the node from the root node. Whenever the index node is full, the marking height is updated one by one. On the leaf, an X latch is acquired. If the leaf node is not full, the updater updates it. If the leaf node is full, a split occurs in the leaf, this latch is released immediately and then the procedure shifts to the second phase.

The second phase: The height of the tree is marked as in the first phase. MARK-OPT tries to acquire the X mode latches on the leaf node and the index node below the height marked in the previous phase. If the nodes involved by SMOs are not protected by X latches, it releases all latches and restarts. This process continues until all the nodes involved by SMOs are protected by X latches.

More precisely, the level of node (h) is one for the root node, and H for the leaf node where H is the height of the tree. When l denotes the level of the Btree where the MARK-OPT has to start to use X latches, the variable l is initially set to H. The height marked during a traverse is denoted by m. The MARK-OPT protocol is shown in Figure 2.

Because MARK-OPT decided the range of the X latch, based on the state of the previous phase obtained by marking, it may require multiple restarts when SMOs have spread. However, the number of maximum phases in the INC-MARK-OPT is H at most as for INC-OPT. MARK-OPT requires only a restart once in many cases because SMOs rarely spread. Therefore, MARK-OPT does not require many restarts like INC-OPT even when SMOs occur on nodes at upper levels.

The MARK-OPT protocol satisfies Condition 1 in 2.2. The reasons are:

1. It is deadlock free because it acquires latches top-down.
2. It does not latch the index nodes with the S, SIX modes, and does not acquire needless X mode latches on nodes not relating to SMOs.

The first phase

```
l := H;
Parent := null; Child := ROOT; h := 1; m := 1;
while h < l do begin
  IX latch on Child, unlatch Parent;
  if Child is not full then
    m := h;        /* Marking */
  Determine NewChild;
  Parent := Child; Child := NewChild; h := h + 1;
end;
X latch on Child and its copies;
Unlatch Parent;
if Child is full then
  Release all granted latches; l := m;
else begin
  Update;
  Release all granted latches; operation completion;
end;
```

The second phase

```
Parent := null; Child := ROOT; h := 1; m := 1;
while h < l do begin
  IX latch on Child, unlatch Parent;
  if Child is not full then
    m := h;        /* Marking */
  Determine NewChild;
  Parent := Child; Child := NewChild; h := h + 1;
end;
X latch on Child and its copies;
Unlatch Parent;
Determine NewChild;
Parent := Child; Child := NewChild; h := h + 1;
while h ≤ H do begin
  X latch on Child and its copies;
  Determine NewChild;
  Parent := Child; Child := NewChild; h := h + 1;
end;
if X latch are not sufficient for SMOs then
  Release all granted latches; l := m; goto 18;
else begin
  Update including the SMOs;
  Release all granted latches; operation completion;
end;
```

Figure 2. The MARK-OPT protocol

3. It never uses a tree latch.

It is easy to prove that the MARK-OPT satisfies the physical consistency requirement for Btrees. When an updater realizes that it does not acquire all required X latches for the SMOs, the updater releases all the latches without modifying any data. Thus, the MARK-OPT essentially follows the two phase locking (2PL). The 2PL ensures the physical consistency of the Btree structure for each update [2].

3.2 Extensions of the MARK-OPT

We propose three variations of the concurrency control protocol, INC-MARK-OPT, 2P-INT-MARK-OPT and 2P-REP-MARK-OPT, which are extensions of MARK-OPT.

In each protocol only the second phase of the update protocol is different. MARK-OPT does not change the process even if the tree structure is changed by other transactions. On the other hand, the extension protocols look at the state of the node first latched with the X mode in that phase and checks the change from the previous phase of a subtree relating to SMO. If the extension protocols judge that the tree structure has been changed, each protocol executes a different process. The difference in the processes is shown in Table 2. The columns indicate the process phases, because

Table 2. Comparison of protocols by handling for a tree structure change between phases

	continue the 2nd phase	shift to the 1st phase
non-restart	MARK-OPT / INC-OPT	2P-INT-MARK-OPT
restart	INC-MARK-OPT	2P-REP-MARK-OPT

the protocols judges if the tree structure has changed, and the rows indicate the presence of a restart at that time.

Because these extension protocols mark the height of the tree as does MARK-OPT, they execute very similar process as MARK-OPT except for phase change. All of these also satisfy Condition 1.

3.2.1 INC-MARK-OPT Protocol

The *incremental marking optimistic (INC-MARK-OPT) protocol* restarts when it judges that the tree structure has changed in the second phase. In this case, the height marked for next phase is not complete because traverse does not reach a leaf node. However, the INC-MARK-OPT decides the range of the X latch based on that information.

The second phase of the INC-MARK-OPT protocol for update is as follows. The INC-MARK-OPT acquires the X mode latches on the node marked in the previous phase. If the node is not full, it executes a process similar to the MARK-OPT to the leaf node. If the node is full, it releases all latches and restarts. This process continues until all the nodes involved by SMOs are protected by X latches.

If the INC-MARK-OPT judges that the tree structure has been changed, it restarts at once. Therefore, the INC-MARK-OPT does not acquire more needless X latches than MARK-OPT when SMOs have actually spread. On the other hand, the INC-MARK-OPT may judge that the tree structure has changed when SMOs have contracted. In that case, the INC-MARK-OPT acquires more needless X latches than MARK-OPT. Moreover, INC-MARK-OPT increases the frequency of restarts compared with MARK-OPT. However, the number of maximum phases in the INC-MARK-OPT is H at most the same as MARK-OPT because it spreads the range of the X latch at each restart.

3.2.2 2P-INT-MARK-OPT Protocol

The *2-phase integrated marking optimistic (2P-INT-MARK-OPT) protocol* performs the latch-coupling with IX latches below the node when it judges that the tree structure has changed in the second phase. That is, it returns to the first phase. Because the MARK-OPT marks the tree state in the second phase, it shifts to the first phase without the problem of marking that is incomplete.

The second phase of the 2P-INT-MARK-OPT protocol for update is as follows. The 2P-INT-MARK-OPT acquires the X mode latches on the node marked in the previous phase. If the node is not full, it executes a process similar to MARK-OPT to the leaf node. If the node is full, it shifts to the first phase on the node.

If the 2P-INT-MARK-OPT judges that the tree structure has changed, it shifts to the first phase. Therefore, the 2P-INT-MARK-OPT decreases the frequency of restarts to a greater extent than the INC-MARK-OPT and it does not acquire more needless X latches than MARK-OPT. Moreover, 2P-INT-MARK-OPT does not acquire needless X latches when SMOs have contracted. On the other hand, the 2P-INT-MARK-OPT may require more restarts than H, the height of the tree. However, this will happen infrequently. In the worst case, the 2P-INT-MARK-OPT cannot complete the process but the 2P-INT-MARK-OPT requires only a small number of restarts because this will not happen in practice.

3.2.3 2P-REP-MARK-OPT Protocol

The *2-phase repetitive marking optimistic (2P-REP-MARK-OPT) protocol* restarts when it judges that the tree structure has changed in the second phase. The 2P-REP-MARK-OPT returns to the first phase after it restarts.

The second phase of the 2P-REP-MARK-OPT protocol for update is as follows. The 2P-REP-MARK-OPT acquires the X mode latch on the node marked in the previous phase. If the node is not full, it executes a process similar to MARK-OPT to the leaf node. If the node is full, it releases all latches and it executes the process in the first phase on the root node.

If the 2P-REP-MARK-OPT judges that the tree structure has changed, it restarts at once and shifts to the first phase. Therefore, the 2P-REP-MARK-OPT acquires the least needless X latches of the proposed protocols although it increases the frequency of restarts more than any of the proposed protocols. As well as 2P-INT-MARK-OPT, the 2P-REP-MARK-OPT may require more restarts than H, the height of the tree. However, the 2P-REP-MARK-OPT requires only a small number of restarts because this is unlikely to happen in practice.

4 Experiments

To show that MARK-OPT and its extension protocols are effective, we implemented them on an autonomous disk system [11] on a blade system, which uses the Fat-Btree, and evaluated their performance under a number of conditions.

Table 3. Experimental Environment

No. of Nodes:	128 (Storages), 16 (Clients)
CPU:	AMD Athlon XP-M 1800+ (1.53GHz)
Memory:	PC2100 DDR SDRAM 1GB
Hard Drive:	TOSHIBA MK3019GAX (30GB, 5400rpm, 2.5inch)
OS:	Linux 2.4.20
Java VM:	Sun J2SE SDK 1.5.0_01 Server VM

Table 4. Parameters used for the experiments

Page size:	4KB
Tuple size:	3KB
Max No. of entries in an index node (fanout):	8
Max No. of tuples in a leaf node:	1

4.1 Experimental Environment

We used an experimental system of the autonomous disk distributed storage technology we proposed. The experimental system was implemented on a 144 node blade system using the Java programming language on Linux. We used 128 nodes for storing data and 16 nodes as clients sending requests. A preliminary experiment showed that the backbone network switch had adequate performance. The experimental environment is summarized in Table 3. Table 4 shows the basic parameters we set for the experiments. These parameters were chosen to distinguish clearly the differences between the protocols.

4.2 Experimental Result

Sixteen clients (thirty-two threads in parallel per blade) sent requests to PEs containing the Fat-Btree with 256 tuples per PE, for 60 seconds. The access frequencies were uniform. Figure 3 shows the performance of the six concurrency controls as the update ratio changes from 0% through 100%. The solid lines show the performance of the four proposed protocols, the dotted line shows the performance of INC-OPT and the dashed line shows the performance of ARIES/IM. The horizontal and vertical axes are the update ratio and throughput, respectively.

When the update ratio was 0%, the results of all protocols were virtually the same. This is because the concurrency controls used to retrieve data are basically the same. But, the throughput of ARIES/IM decreases sharply even though the increase in the ratio of update operations is small. This is because the cost of global synchronization by using the tree latches for SMOs caused by the update operations. On the other hand, when the update ratio is low,

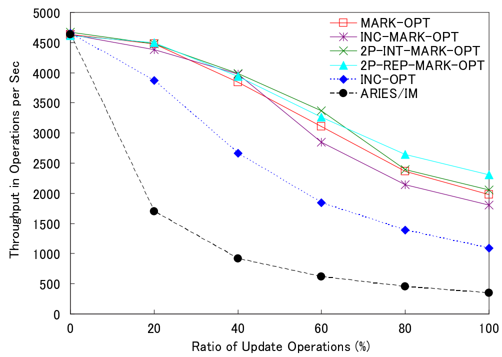

Figure 3. Comparison of concurrency control protocols with changing update ratio

the results of all other protocols were better than that of ARIES/IM and almost the same. However, the throughput of the INC-OPT decreases as the update ratio increases. In contrast to the INC-OPT, the proposed protocols can provide reasonable throughput although the update ratio increases. The decline in the throughput of the proposed protocols is much slower than that of INC-OPT even when the update operations are included. This is because the proposed protocols reduce the frequency of restarts compared with the INC-OPT when SMOs occur, although the increase in the update ratio increases the occurrence of SMOs.

In the comparison of the proposed protocols, the throughput of 2P-REP-MARK-OPT is the highest, when the update ratio is 100%. Then, the throughput of 2P-INT-MARK-OPT is next highest and then MARK-OPT is higher than INC-MARK-OPT. The throughput of the INC-MARK-OPT is the lowest. Therefore, the 2P-REP-MARK-OPT is the most effective when many SMOs occur.

5 Conclusion

We propose a new concurrency control, MARK-OPT, for parallel Btrees, for the shared-nothing environment. When SMOs occur, the proposed protocol marks the node for which the X latch should be acquired first and it acquires the X latch nodes below the marked height after it restarts. We also propose three extensions of the MARK-OPT protocol. In addition, we have experimented on an autonomous disk system implemented on a large-scale blade system to compare the four proposed protocols and the conventional protocols. The experimental results indicated that the proposed protocols were effective and 2P-REP-MARK-OPT was the superior protocol.

In future studies we plan to apply the B-link [7, 6] to the Fat-Btree. It is known that the B-link can achieve excellent concurrency control. The B-link uses links to chain all nodes at each level together. In the B-link algorithm, neither readers nor updaters latch-couple on their way down to a leaf node and they acquire the latch only on one node at a time. Moreover, the B-link algorithm does not require restarts and they complete processing during one traverse. We need to examine how to apply the B-link to the Fat-Btree and compare this and the conventional protocols.

Acknowledgments

We thank Dr. Jun Miyazaki of NAIST for his advice on concurrency control for the Fat-Btree. This work is partially supported by CREST of JST (Japan Science and Technology Agency), SRC (Storage Research Consortium), a Grant-in-Aid for Scientific Research from MEXT Japan (#16016232) and the Tokyo Institute of Technology 21COE Program "Framework for Systematization and Application of Large-Scale Knowledge Resources".

References

[1] R. Bayer and M. Schkolnick. Concurrency of Operations on B-trees. *Acta Inf.*, 9(1):1–21, 1977.

[2] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[3] G. Copeland, W. Alexander, E. Boughter, and T. Keller. Data Placement in Bubba. In *Proc. of ACM SIGMOD'88*, pages 99–108, 1988.

[4] S. Ghandeharizadeh and D. J. DeWitt. Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines. In *Proc. of VLDB'90*, pages 481–492, 1990.

[5] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1992.

[6] V. Lanin and D. Shasha. A Symmetric Concurrent B-tree Algorithm. In *Proc. of FJCC'86*, pages 380–389, 1986.

[7] P. L. Lehman and S. B. Yao. Efficient Locking for Concurrent Operations on B-trees. *ACM Trans. Database Syst.*, 6(4):650–670, 1981.

[8] J. Miyazaki and H. Yokota. Concurrency Control and Performance Evaluation of Parallel B-tree Structures. *IEICE Trans. Inf. Syst.*, E85-D(8):1269–1283, 2002.

[9] C. Mohan and F. Levine. ARIES/IM: An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging. In *Proc. of ACM SIGMOD'92*, pages 371–381, 1992.

[10] V. Srinivasan and M. J. Carey. Performance of B-Tree Concurrency Control Algorithms. In *Proc. of ACM SIGMOD'91*, pages 416–425, 1991.

[11] H. Yokota. Autonomous Disks for Advanced Database Applications. In *Proc. of DANTE'99*, pages 435–442, 1999.

[12] H. Yokota, Y. Kanemasa, and J. Miyazaki. Fat-Btree: An Update-Conscious Parallel Directory Structure. In *Proc. of ICDE'99*, pages 448–457, 1999.