PAPER

MARK-OPT: A Concurrency Control Protocol for Parallel B-Tree Structures to Reduce the Cost of SMOs[*]

Tomohiro YOSHIHARA[†a)], **Dai KOBAYASHI**[†,††], ***Nonmembers***, ***and*** **Haruo YOKOTA**[†,†††], ***Member***

SUMMARY In this paper, we propose a new concurrency control protocol for parallel B-tree structures capable reducing the cost of structure-modification-operation (SMO) compared to the conventional protocols such as ARIES/IM and INC-OPT. We call this protocol the MARK-OPT protocol, since it marks the lowest SMO occurrence point during optimistic latch-coupling operations. The marking reduces middle phases for spreading an X latch and removes needless X latches. In addition, we propose three variations of the MARK-OPT, which focus on tree structure changes from other transactions. Moreover, the proposed protocols are deadlock-free and satisfy the physical consistency requirement for B-trees. These indicate that the proposed protocols are suitable as concurrency control protocols for B-tree structures. To compare the performance of the proposed protocols, the INC-OPT, and the ARIES/IM, we implement these protocols on an autonomous disk system adopting the Fat-Btree structure, a form of parallel B-tree structure. Experimental results in various environments indicate that the proposed protocols always improve system throughput, and 2P-REP-MARK-OPT is the most useful protocol in high update environment. Additionally, to mitigate access skew, data should be migrated between PEs. We also demonstrate that MARK-OPT improves the system throughput under the data migration and reduces the time for data migration to balance load distribution.

key words: *index, concurrency control, B-tree, parallel DB, latch*

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 [1], [2]. The value range partitioning method with a parallel B-tree 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 B-tree 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 B-tree, the synchronization between the PEs degrades the system throughput considerably. On the other hand, if all index nodes of a B-tree 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 B-tree structure, Fat-Btree, has been proposed, and experimental results indicate that Fat-Btrees provide better performance than other parallel B-trees [3].

It is also important to provide an efficient concurrency control protocol for parallel B-tree structures, including Fat-Btree. The INC-OPT protocol, which is suitable for a parallel B-tree on a shared-nothing parallel machine, has been proposed [4]. The INC-OPT protocol outperforms the conventional B-tree concurrency control protocols, such as the B-OPT protocol [5] and the ARIES/IM protocol [6]. 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 B-tree 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-OPT and the ARIES/IM[**] on an autonomous disk system [7] using the Fat-Btree as the distributed directory structure and measured the system throughput with changing update ratio, the number of threads on the clients. These experimental results indicate that the proposed protocols are effective.

To mitigate access skew, data should be migrated between PEs. We also compared the throughput of MARK-OPT with INC-OPT and ARIES/IM with changing patterns of data migration. The experimental results indicate that the MARK-OPT is also effective with data migration.

The concurrency control methods for update operations and data migrations with the parallel B-trees are important not only for parallel database systems, but also for more

Manuscript received May 2, 2006.
Manuscript revised January 4, 2007.

[†]The authors are with the Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology, Tokyo, 152–8552 Japan.
[††]The author is also a JSPS Research Fellow (DC).
[†††]The author is also with the Global Scientific Information and Computing Center, Tokyo Institute of Technology, Tokyo, 152–8550 Japan.

[*]Preliminary versions of parts of this paper appear in the Proceedings of the 16th Data Engineering Workshop of IEICE (DEWS2005), DBSJ Letters, and the Proceedings of IEEE SWOD2006.

a) E-mail: yoshihara@de.cs.titech.ac.jp

DOI: 10.1093/ietisy/e90–d.8.1213

[**]We implemented ARIES/IM based on [6], but did not recovery mechanism of it because we do not focus on recovery in this paper.

general parallel data storage. The number of unstructured data files is increasing rapidly so they should be stored on some parallel data storage having a global name space to manage them efficiently. To realize efficient management functions, including skew handling for a large number of globally named files, an effective parallel B-tree with sophisticated concurrency control is required.

The remainder of the paper is organized as follows. First, the concept of the Fat-Btree and data migration are introduced as background in Sect. 2. Section 3 describes the concurrency controls for parallel B-trees. We propose the new concurrency control protocols for parallel B-tree structures in Sect. 4. The experimental results are reported in Sect. 5. We review related work in Sect. 6. The final section presents the conclusions of this paper.

2. Background

2.1 Fat-Btree Structure

The Fat-Btree [3] is a form of parallel B-tree in which the leaf pages of the B^+-tree are distributed among PEs and each PE has a subtree of the whole B-tree, 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 B-tree structure.

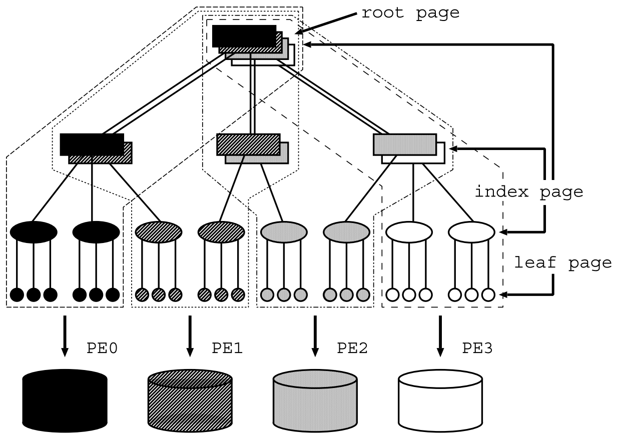

Fig. 1 Fat-Btree.

2.2 Data Migration with a Fat-Btree

Data migration is effective in handling the skew of an access request distribution. For range partitioned data placement the total access frequency of data stored in each PE is eventually balanced by migrating data between PEs logically adjacent in the range partition.

The outline of the algorithm for migrating nodes is described in [3]. Figure 2 illustrates an example of the data migration with a Fat-Btree, where the right-most side leaf index page with data pages in a PE are migrated to its right side PE. In this case, because only two consecutive PEs are involved in the data migration, the data migration can be achieved without blocking the processes of the other PEs.

The tree structure of a Fat-Btree is unchanged by data migration. However, the migration of a data page causes an update in its parent index node and, occasionally in more remote ancestor nodes, by recursive update. If the parent node already exists in the destination PE, only an update of the corresponding entry is required; otherwise, the parent node must be moved to the destination PE. If the source PE contains other children of the parent node, the source PE must retain a copy of the node; otherwise the node in the source PE must be removed.

3. Concurrency Control Methods

Some concurrency control method for the B-tree is necessary to guarantee its consistency. The concurrency control for handling SMOs influences the throughput. And that SMOs on the parallel B-trees may require communicating across a network to synchronize pages on each PE Therefore, the concurrency control for the parallel B-trees structures is especially important.

Instead of locks, fast and simple latches are usually used for concurrency control during traversing index nodes in a B-tree [8]. A latch is a form of semaphore and the latch manager does not have a deadlock detection mechanism. Therefore, concurrency control for a B-tree node should be deadlock-free.

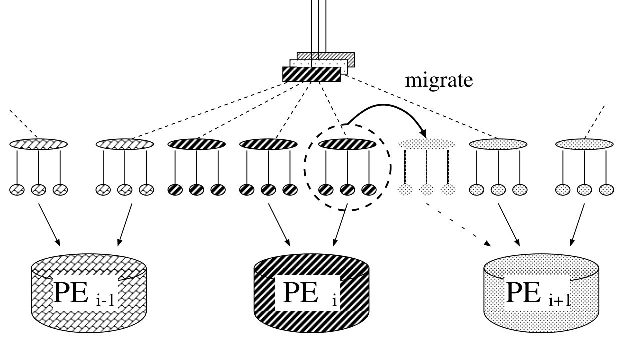

Fig. 2 Data migration in Fat-Btree.

3.1 Latch Modes

In this paper, a latch is assumed to have five modes: IS, IX, S, SIX, and X as shown in Table 1 [8]. The symbol of "○" means that the two modes are compatible.

Because a parallel B-tree 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 deadlock. This means synchronization cost grows in proportion to the number of PEs related to latches.

3.2 Concurrency Control for a Fat-Btree

As described above, concurrency control of the access path should be deadlock-free because the latch which does not have deadlock detection mechanism is effective for the B-tree.

Figure 3 shows the Fat-Btree on which duplicating pages in Fig. 1 is eliminated. Only two X latches must be acquired on PE0 if an SMO occurs at pages bounded by the dashed line at the lower left in Fig. 3. X latches must be acquired on PE2 and PE3 at a time to synchronize copies of index page on PE2 and PE3 if an SMO occurs at pages across PEs bounded by the dotted line at the lower right in Fig. 3. Therefore, the SMO like this requires communicating across a network. X latches must be acquired on all PEs

Table 1 Latch matrix.

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

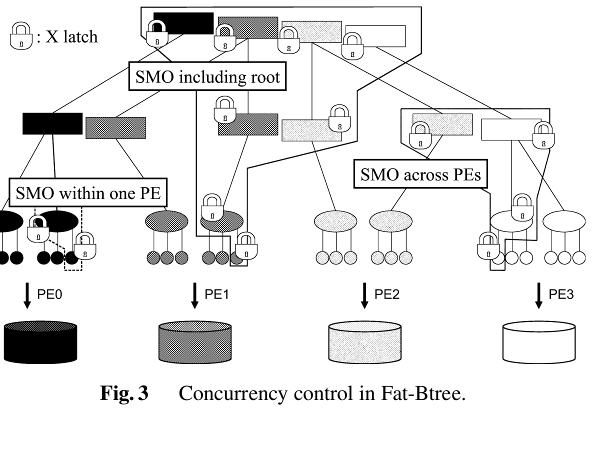

Fig. 3 Concurrency control in Fat-Btree.

at a time if an SMO occurs at pages including root pages bounded by the solid line at the center of Fig. 3. The cost of acquiring X latches on pages at upper level is larger like this. Because the S, SIX and X modes in a parallel B-tree require synchronization between PEs and use of these latches on nodes with the copies in many PEs increases synchronous overheads between PEs. Therefore, on index nodes at upper levels with the copies in many PEs in the Fat-Btree use of the S, SIX, and X mode latches should be avoided as much as possible except when the X latches at the root pages have no other choice to be acquired to update the root pages.

An alternative concurrency control protocol, suggested by Mohan et al. [6], acquires an X tree latch to protect the entire B-tree when SMOs occur. This protocol in a parallel B-tree in a distributed environment is simplified if a specified PE takes care of the tree latch. However, the PE taking care of the tree latch becomes a bottleneck when the number of PEs increases. The bottleneck prevents higher throughput by more PEs in the distributed environment. Moreover, the synchronous overhead between PEs is large because latches on the entire tree are acquired. Communication to acquire the tree latch is required even when an SMO frequently occurs within one PE. Therefore, the concurrency control using the tree latch is unsuitable for the parallel B-tree.

As a summary of the conditions above, the following conditions of the concurrency controls for parallel B-trees are proposed in [4].

Condition 1: A concurrency control method for parallel B-trees 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 B-tree should be avoided as much as possible.
(c) The entire tree should not be latched, even for a short duration.

B-OPT [5], OPT-DLOCK [9], and ARIES/IM [6] are excellent concurrency control methods for a B-tree 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 B-trees, such as Fat-Btree.

3.3 INC-OPT Protocol

The INC-OPT protocol satisfying Condition 1 has been proposed [4] as a concurrency control protocol for parallel B-trees, 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 B-tree:

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 [8]. 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 [4].

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 B-tree. 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.

4. Proposed Protocols

4.1 MARK-OPT Protocol

We propose a new concurrency control protocol for parallel B-trees, 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 not full, MARK-OPT marks the height of the node from the root node. Whenever the index node is not 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 is H for the leaf node where H is the height of the tree. When l denotes the level of the B-tree where the MARK-OPT has to start to use X latches, the variable l is initially set to H, thus, only a leaf node is latched with the X mode in the first phase like INC-OPT. The height marked during a traverse is denoted by m. The MARK-OPT protocol is shown in Fig. 4. The first phase and the second phase in Fig. 4 show the same in the previous paragraph. During the latch-coupling on lines 3 to 9 or lines 19 to 25 in Fig. 4, the marking of MARK-OPT is performed on line 6 or 22. The variable l is set to m on line 13 or 36 before restarts, thus, the leaf node and the index node below the height (m) marked in the previous phase is latched with the X mode in lines 26 to 34. When all the nodes involved by SMOs are

```
The first phase
 1  l := H;
 2  Parent := null; Child := ROOT; h := 1; m := 1;
 3  while h < l do begin              /* Latch-coupling */
 4      IX latch on Child, unlatch Parent;
 5      if Child is not full then
 6          m := h;                   /* Marking */
 7      Determine NewChild;
 8      Parent := Child; Child := NewChild; h := h + 1;
 9  end;
10  X latch on Child and its copies;
                                      /* Leaf is X latched */
11  Unlatch Parent;
12  if Child is full then
13      Release all granted latches; l := m;
                                      /* Restart */
14  else begin
15      Update;
16      Release all granted latches; operation completion;
17  end;
The second phase
18  Parent := null; Child := ROOT; h := 1; m := 1;
19  while h < l do begin              /* Latch-coupling */
20      IX latch on Child, unlatch Parent;
21      if Child is not full then
22          m := h;                   /* Marking */
23      Determine NewChild;
24      Parent := Child; Child := NewChild; h := h + 1;
25  end;
26  X latch on Child and its copies;
                      /* Index on marking is X latched */
27  Unlatch Parent;
28  Determine NewChild;
29  Parent := Child; Child := NewChild; h := h + 1;
30  while h ≤ H do begin
31      X latch on Child and its copies;
                      /* Nodes below marking are X latched */
32      Determine NewChild;
33      Parent := Child; Child := NewChild; h := h + 1;
34  end;
35  if X latch are not sufficient for SMOs then
36      Release all granted latches; l := m; goto 18;
                                      /* Restart */
37  else begin
38      Update including the SMOs;
39      Release all granted latches; operation completion;
40  end;
```

Fig. 4 The MARK-OPT protocol.

protected by X latches, the update on line 38 is performed.

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 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 Sect. 3.2. 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.
3. It never uses a tree latch.

It is easy to prove that the MARK-OPT satisfies the physical consistency requirement for B-trees. 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 B-tree structure for each update [10].

4.2 Extensions of the MARK-OPT Protocol

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. When the marked node in the first phase is updated from other transactions, the mark may not show the range of the SMO. It is not effective to acquire X latches based on the mark like MARK-OPT at such time. These extension protocols focus on tree structure changes from other transactions. To process more effectively than MARK-OPT when the marked node is changed, we propose these protocols which reconsider the range of the SMO at such time.

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 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.

Table 2 Comparison of concurrency control 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

4.2.1 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.

We indicate an example that MARK-OPT processes inefficiently. The range of X latches acquired according to the mark is not enough, if the nodes below the height of marking became full by modification of other transaction while MARK-OPT restarts. In this time, MARK-OPT also restarts after it traverses to the leaf node with acquiring X latches, and it has useless traversal and X latches. To not have useless traversal and latches like this, INC-MARK-OPT judges that the tree structure has changed, it does not traverse to the leaf node and it restarts on the node. Then, it acquires the X latches according to renewed mark. Therefore, in this case, INC-MARK-OPT acquires less X latches than MARK-OPT, and INC-MARK-OPT processes more efficiently than MARK-OPT.

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.

We define the INC-MARK-OPT protocol precisely by using the same variables in Fig. 4. The INC-MARK-OPT protocol is shown in Fig. 5. The first phase is omitted since this phase is the same in Fig. 4. The part in the frame is added to MARK-OPT. In this part, INC-MARK-OPT checks the marked node change and performs handling for the change.

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.

```
The second phase
18  Parent := null; Child := ROOT; $h$ := 1; $m$ := 1;
19  while $h < l$ do begin            /* Latch-coupling */
20      IX latch on Child, unlatch Parent;
21      if Child isn't full then
22          $m := h$;          /* Marking */
23      Determine NewChild;
24      Parent := Child; Child := NewChild; $h := h + 1$;
25  end;
26  X latch on Child and its copies;
                        /* Index on marking is X latched */
27  Unlatch Parent;
28  if Child is full then
29      Release all granted latches; $l := m$; goto 18;
                                             /* Restart */
30  Determine NewChild;
31  Parent := Child; Child := NewChild; $h := h + 1$;
32  while $h \leq H$ do begin
33      X latch on Child and its copies;
                  /* Nodes below marking are X latched */
34      Determine NewChild;
35      Parent := Child; Child := NewChild; $h := h + 1$;
36  end;
37  if X latch are not sufficient for SMOs then
38      Release all granted latches; $l := m$; goto 18;
                                             /* Restart */
39  else begin
40      Update including the SMOs;
41      Release all granted latches; operation completion;
42  end;
```

Fig. 5 The INC-MARK-OPT protocol.

```
The second phase
18  Parent := null; Child := ROOT; $h$ := 1; $m$ := 1;
19  while $h < l$ do begin            /* Latch-coupling */
20      IX latch on Child, unlatch Parent;
21      if Child isn't full then
22          $m := h$;          /* Marking */
23      Determine NewChild;
24      Parent := Child; Child := NewChild; $h := h + 1$;
25  end;
26  X latch on Child and its copies;
                        /* Index on marking is X latched */
27  Unlatch Parent;
28  if Child is full then
29      $l := H$; goto 7;
                               /* Shift to the first phase */
30  Determine NewChild;
31  Parent := Child; Child := NewChild; $h := h + 1$;
32  while $h \leq H$ do begin
33      X latch on Child and its copies;
                  /* Nodes below marking are X latched */
34      Determine NewChild;
35      Parent := Child; Child := NewChild; $h := h + 1$;
36  end;
37  if X latch are not sufficient for SMOs then
38      Release all granted latches; $l := m$; goto 18;
                                             /* Restart */
39  else begin
40      Update including the SMOs;
41      Release all granted latches; operation completion;
42  end;
```

Fig. 6 The 2P-INT-MARK-OPT protocol.

4.2.2 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.

On the example in Sect. 4.2.1, 2P-INT-MARK-OPT judges that the tree structure has changed, it acquires the X latches according to renewed mark after it traverse to the leaf node with the latch-coupling with the IX latches as well as first traversal. The latch-coupling with IX latches during second traversal of 2P-INT-MARK-OPT has lower cost than acquiring X latches during that of MARK-OPT. Therefore, 2P-INT-MARK-OPT processes more efficiently than MARK-OPT. In this case, 2P-INT-MARK-OPT processes obviously less efficiently than INC-MARK-OPT because of useless second traversal with latch-coupling. However, if the node on the height of marking is full but the nodes below the height are not full, 2P-INT-MARK-OPT performs the update in second traversal. On the other hand, INC-MARK-OPT performs the update in third traversal. So, two traversals of 2P-INT-MARK-OPT are more efficient than three traversals of INC-MARK-OPT.

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.

We define the 2P-INT-MARK-OPT protocol precisely by using the same variables in Fig. 4. The 2P-INT-MARK-OPT protocol is shown in Fig. 6. The first phase is omitted since this phase is the same in Fig. 4. The part in the frame is added to MARK-OPT. In this part, 2P-INT-MARK-OPT checks the marked node change and performs handling for the change.

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.

4.2.3 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.

Because 2P-INT-MARK-OPT performs the latch-coupling with IX latches after it judges that the tree structure has changed, with acquiring X latches, it may wait for other transactions to release latches. This decreases the degree of parallel operations. On the other hand, in this case, 2P-REP-MARK-OPT releases all granted latches and it restarts. As above, 2P-REP-MARK-OPT does not wait with acquiring X latches except acquiring for updates in last traversal. Therefore, 2P-REP-MARK-OPT does not wait with acquiring useless X latches.

The second phase
```
18  Parent := null; Child := ROOT; $h$ := 1; $m$ := 1;
19  while $h$ < $l$ do begin            /* Latch-coupling */
20      IX latch on Child, unlatch Parent;
21      if Child isn't full then
22          $m$ := $h$;           /* Marking */
23      Determine NewChild;
24      Parent := Child; Child := NewChild; $h$ := $h$ + 1;
25  end;
26  X latch on Child and its copies;
                        /* Index on marking is X latched */
27  Unlatch Parent;
28  if Child is full then
29      Release all granted latches; goto 1;
                  /* Restart and shift to the first phase */
30  Determine NewChild;
31  Parent := Child; Child := NewChild; $h$ := $h$ + 1;
32  while $h$ ≤ $H$ do begin
33      X latch on Child and its copies;
                  /* Nodes below marking are X latched */
34      Determine NewChild;
35      Parent := Child; Child := NewChild; $h$ := $h$ + 1;
36  end;
37  if X latch are not sufficient for SMOs then
38      Release all granted latches; $l$ := $m$; goto 18;
                                          /* Restart */
39  else begin
40      Update including the SMOs;
41      Release all granted latches; operation completion;
42  end;
```

Fig. 7 The 2P-REP-MARK-OPT protocol.

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.

We define the 2P-REP-MARK-OPT protocol precisely by using the same variables in Fig. 4. The 2P-REP-MARK-OPT protocol is shown in Fig. 7. The first phase is omitted since this phase is the same in Fig. 4. The part in the frame is added to MARK-OPT. In this part, 2P-REP-MARK-OPT checks the marked node change and performs handling for the change.

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.

4.3 Comparisons of Proposed Protocols

We compare the features of the protocols for update. The X latch acquisition time per process is obtained by the frequency of restarts and the X latch acquisition time per phase. This time influences the performance of the protocol. Therefore, we compare the frequency of restarts and the X latch acquisition time per phase of the proposed protocols.

First, we compare the frequency of restarts of each of the proposed protocols. The extension protocols increase the frequency of restarts more than MARK-OPT, and 2P-REP-MARK-OPT increases the frequency of restarts in the proposed protocols more than the others.

Next, we compare the X latch acquisition time per phase for each of the proposed protocols. At the same time, the extension protocols do not acquire X latches on multiple nodes apart from the update phase. Therefore, the extension protocols decrease the X latch acquisition time per phase more than the MARK-OPT. The INC-MARK-OPT increases the X latch acquisition time per phase more than the other extension protocols because it increases the acquisition of the X latches on a node in the upper levels more than the others.

We compare these proposed protocols experimentally in Sect. 5.5.

5. Experiments

To show that MARK-OPT and its extension protocols are effective, we implemented them on an autonomous disk system [7], [11], [12] on a blade system, which uses the Fat-Btree, and evaluated their performance under a number of conditions.

5.1 Experimental Environment

We used an experimental system of the autonomous disk distributed storage technology we proposed [7]. 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, which does not limit the system throughput in the experiment. The experimental environment is summarized in Table 3.

Table 3 Experimental environment.

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

5.1.1 Initial Fat-Btree Construction

We prepared a leaf node on each PE, a seed of the Fat-Btree and the key of the leaf node was the lower bound value of the key of node stored in the PE. We set the key value of the initial leaf nodes in ascending order of PE number. Therefore, the leaf node stored in each PE by follow-on insertion was divided statically. We then repeatedly inserted random elements in the initial Fat-Btree. Table 4 shows the basic parameters we set for the experiments. These parameters were chosen to distinguish clearly the differences between the protocols.

Table 4 Parameters used for the experiments.

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

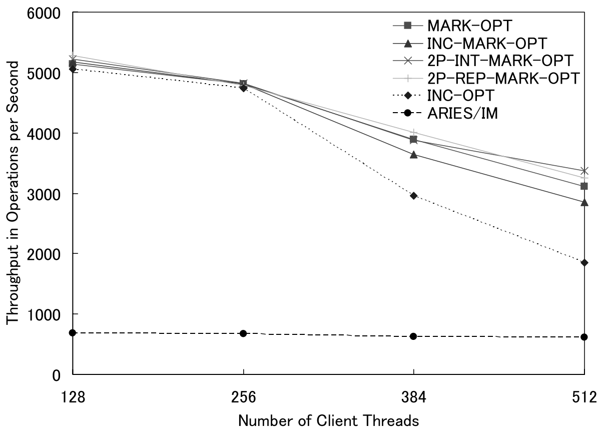

Fig. 8 Comparison of concurrency control protocols with changing numbers of threads on clients.

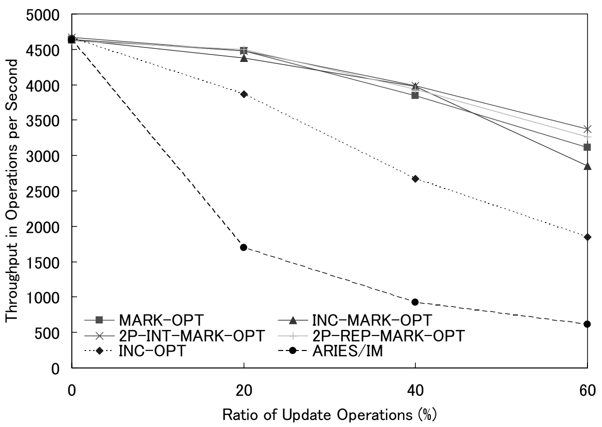

Fig. 9 Comparison of concurrency control protocols with changing update ratio.

The experiments for this paper were organized as follows. First, we evaluated the performance by changing the number of client threads. Next, we evaluated the performance by changing the update ratio, counting the frequency of restarts. Then, we evaluated the performance and counted the frequency of restarts and acquisitions of X latches in a high update environment. Finally, we evaluated the performance with data migrations, measuring the data migration execution time.

5.2 Comparison with Changing Number of Threads on Clients

Sixteen clients sent requests to the PEs containing the Fat-Btree with 256 tuples per PE, for 60 seconds. The access frequencies were uniform and the update ratio was fixed at 60%.

Figure 8 shows the performance of the six concurrency controls when the total number of threads on clients in the whole system changes from 128 through 512 (the number of threads in parallel per blade changes from eight through thirty-two). 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 number of the threads on clients and throughput, respectively.

The throughput of ARIES/IM is always very low. This is because ARIES/IM must synchronize among all the PEs by acquiring the tree latch when an SMO happens. The cost of this synchronization extremely degrades the system performance. Moreover, the throughput of the INC-OPT decreases as the numbers of client threads increases. In contrast to the INC-OPT, the proposed protocols can provide reasonable throughput although the numbers of client threads increases. The decline in the throughput of the proposed protocols is much slower than that of INC-OPT even when the frequency of accesses from the clients is very high. This is because the proposed protocols reduce the frequency of restarts compared with the INC-OPT when SMOs occur, although the increase in the accesses from clients increases the occurrence of SMOs.

When update ratio is less than 60%, the throughput of all protocols increases, and the difference of each throughput contracts. However, the proposed protocols always improve the throughput of the conventional protocols when the update operations are included. This can be inferred from the experimental result in Sect. 5.3.

5.3 Comparison with Changing Update Ratio

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 9 shows the performance of the six concurrency controls as the update ratio changes from 0% through 60%. 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, 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

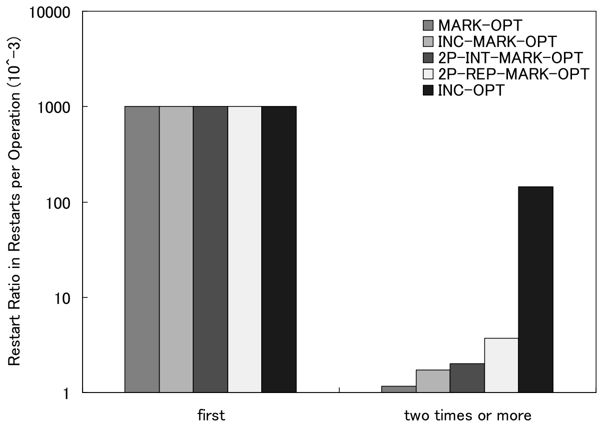

Fig. 10 Comparison of restart ratio (update ratio: 60%).

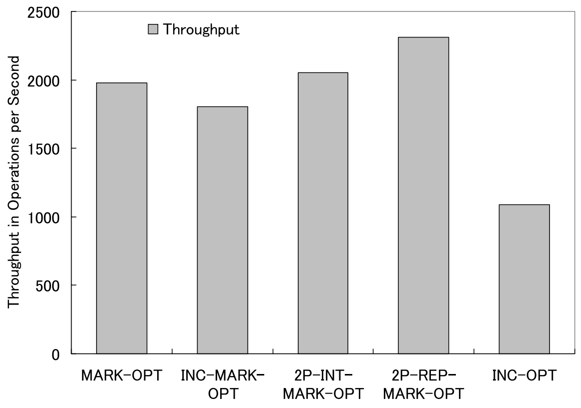

Fig. 11 Comparison of throughput in high update environment (update ratio: 100%).

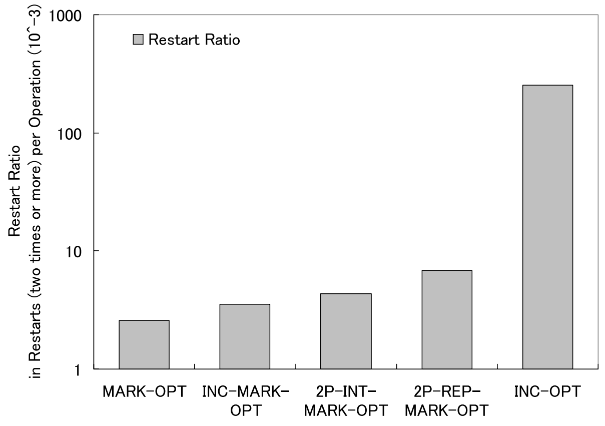

Fig. 12 Comparison of restart in high update environment (update ratio: 100%).

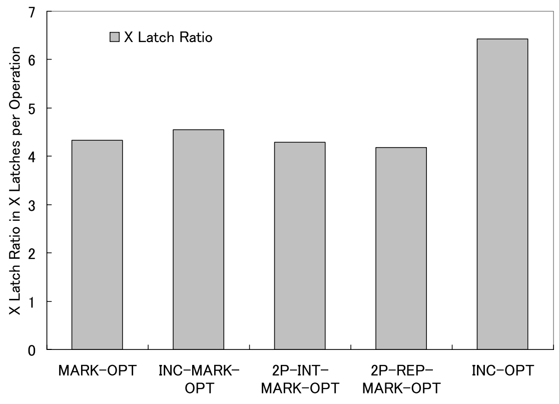

Fig. 13 Comparison of X latch ratio in high update environment (update ratio: 100%).

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. Moreover, the results of the four proposed protocols are always similar.

5.4 Comparison of Restart Ratio

We also counted the frequency of restarts in the experiment described in Sect. 5.3. Figure 10 shows the frequency of restarts per operation of the five concurrency controls when the update ratio is 60%. The left and right charts show the ratio of first restart and second and more restarts on the operation, respectively.

The ratios of first restart on the operation of all protocols are almost the same, because the concurrency controls of an update are basically the same until first restarts. The ratio of second and greater restarts on the operation of the INC-OPT is much higher than that of the proposed protocols, because the INC-OPT requires multiple restarts when SMOs occur extensively. In the comparison of the proposed protocols, the ratio of second and greater restarts on the operation of the 2P-REP-MARK-OPT is the highest, those of INC-MARK-OPT and 2P-INT-MARK-OPT are next, and that of MARK-OPT is the lowest.

This result indicates that marking SMO occurrence point in the proposed protocols is effective. The performances of the proposed protocols are similar because they rarely required multiple restarts in this experiment. We propose the extensions for the environment where many SMOs occur. Therefore, the performances of the proposed protocols are similar in the other environments.

5.5 Comparison in High Update Environment

The performances of the proposed protocols were similar in an environment where many SMOs did not occur. To compare the proposed protocols, we experimented in an unrealistic environment where many SMOs occur. For that purpose, we set the update ratio to 100%.

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. Figures 11, 12, and 13 show the performance of the five concurrency controls, the frequency of second and greater restarts per operation of those, and the frequency of acquisitions of X latches per operation of those, respectively.

The throughput of INC-OPT is lower than that of the proposed protocols as in the other experiments. This is because the INC-OPT requires more numerous multiple restarts than the proposed protocols in this environment.

In the comparison of the proposed protocols, the fewest multiple restarts are for MARK-OPT. MARK-OPT does not restart when the tree structure is changed. The multiple restarts of INC-MARK-OPT and that of 2P-INT-MARK-OPT are next lowest, INC-MARK-OPT restarts at once when the tree structure is changed and 2P-INT-MARK-OPT returns to the first phase although it does not restart. In addition, the multiple restarts of 2P-REP-MARK-OPT are the greatest. 2P-REP-MARK-OPT restarts and returns to the first phase when the tree structure is changed. As above, although Fig. 12 shows the characteristics of the restart ratio of the proposed protocols, it does not show the causations between the restart ratio and the throughput. This is because the proposed protocols perform different processes during second or later each traversal and much traversal do not directly impact the throughput.

On the other hand, the throughput of the proposed protocols is associated with the X latch ratio. Because the proposed protocols rarely perform different processes, the gap of the throughput among the proposed protocols is not wide. INC-MARK-OPT does not improve the throughput of MARK-OPT. This is because it may acquire undue X latches according to the marking based on checking the nodes at upper levels. Therefore, the throughput of the INC-MARK-OPT is the lowest because of the time of acquisition per X latch. On the other hand, 2P-INT-MARK-OPT and 2P-REP-MARK-OPT improve the throughput of MARK-OPT. This is because two protocols require one nodes latched with X mode at a time except acquiring for updates in last traversal and they acquire X latches according to the marking based on checking the nodes on all height in the last traversal. And that 2P-REP-MARK-OPT with acquiring X latches does not wait for other transactions to release latches except acquiring for updates in last traversal although the protocols frequently wait when many SMOs occur. As a result, 2P-REP-MARK-OPT is the most effective when many SMOs occur.

5.6 Comparison with Changing Ratio of Data Migration

To change the frequency of data migrations we changed the number of processes in parallel to migrate the data. Each process migrated the data on different PEs. To migrate repeatedly 100 leaf nodes between two adjacent PEs, the clients sent data migration requests.

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 and the update ratio was fixed at 40%. Figure 14 shows the performance of the two concurrency controls when the number of processes migrating in parallel changes from 0 through 64. The solid and dotted and dashed lines show the

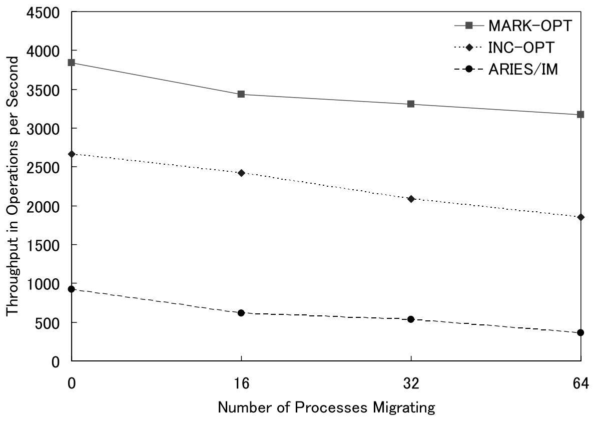

Fig. 14 Comparison with changing ratio of data migration.

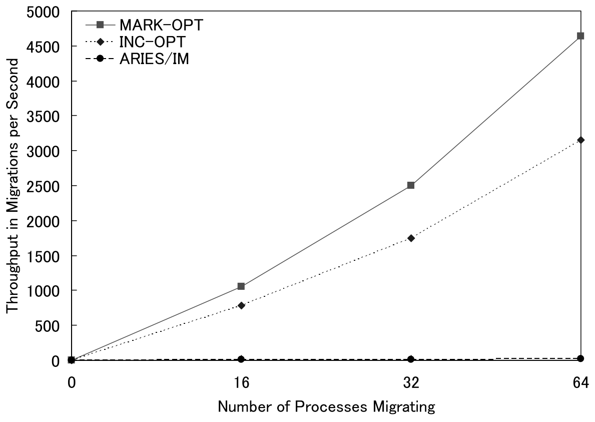

Fig. 15 Comparison of data migration throughput.

performances of MARK-OPT, INC-OPT and ARIES/IM, respectively. The horizontal and vertical axes are the number of processes migrating in parallel and the throughput respectively. Moreover, Fig. 15 shows the throughput of data migration.

The throughput of INC-OPT decreases as the frequency of data migrations increase. On the other hand, MARK-OPT can provide acceptable throughput even when the frequency of data migrations increases. The decline in the throughput of MARK-OPT is much less than that of INC-OPT. In addition, MARK-OPT improves the throughput of data migration. This is because many SMOs occur with increasing data migration. Moreover, ARIES/IM takes quite a lot of time to the data migration. This is because the data migration is sure to cause a SMO.

MARK-OPT can always provide high throughput in an environment with data migration. In addition, MARK-OPT reduces the time for data migration. Therefore, MARK-OPT can achieve fast load-balancing operations. These indicate that MARK-OPT is an effective concurrency control for data migration.

6. Related Work

A concurrency control protocol for shared-nothing parallel

B-trees such as Fat-Btree that is combined INC-OPT with preparatory operation (PO) [13], the preparatory operations-parallel (PO-P) protocol has been developed [14]. In PO-P, POs are adapted for update processes when they encounter unsafe nodes. By using this method any update operations are done in small atomic operations which require only two levels' nodes latched at a time on the global parallel B-tree. Therefore, PO-P increases the degree of parallel operations by decreasing the latch wait time for processes which run in parallel on many PEs.

Although both MARK-OPT and PO-P are the extensions of INC-OPT, they have different approaches for improvement of INT-OPT. MARK-OPT reduces the frequency of restarts by marking the SMO occurrence during optimistic latch coupling operations. The reduction also reduces the frequency of X latches. On the other hand, PO-P reduces the number of nodes on which X latches are acquired at a time. However, PO-P requires the restarts each time unsafe nodes appear, and PO-P does not reduce the frequency of restarts. Therefore, we expect that the response time of MARK-OPT is shorter than that of PO-P. On the other hand, MARK-OPT finally acquires X latches on all nodes relevant to an SMO at a time although PO-P always requires only two levels' nodes latched. Therefore, we think that the degree of parallel operations on MARK-OPT is lower than PO-P. As above, it is difficult to say which protocol is better on the qualitative evaluation. We would like to evaluate the performance of our protocols and PO-P on the experimental system as future work.

7. Conclusion

We propose a new concurrency control, MARK-OPT, for parallel B-trees, 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. These extensions focus on tree structure changes from other transactions. Four proposed protocols are classified according to the presence of the phase shift and the restart processed if the tree structure has changed. 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 are effective and the relationship of frequency of restarts and the time for acquisition of the X latch showed that 2P-REP-MARK-OPT was the superior protocol.

Moreover, we have evaluated the performance of the proposed protocols with the data migration. We have shown that the proposed protocol improves the system throughput with data migration and reduces the time required for data migration. Therefore, the proposed protocol is effective as a concurrency control for data migration. Data migration should be executed based on load evaluation, although clients sent requests for data migration in this paper. An autonomous disk system can autonomously execute these operations. Therefore, it is necessary to experiment with the function.

In future studies we plan to apply the B-link [15], [16] 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 it completes 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.

Moreover, we would like to compare our methods with the PO-P protocol quantitatively by experiments, as described in Sect. 6.

Acknowledgments

The authors would like to thank Dr. Jun Miyazaki of NAIST for his advice on concurrency control for the Fat-Btree. This work was 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), the Tokyo Institute of Technology 21COE Program "Framework for Systematization and Application of Large-Scale Knowledge Resources", and the NHK science and technical research laboratories.

References

[1] G. Copeland, W. Alexander, E. Boughter, and T. Keller, "Data placement in Bubba," Proc. 1988 ACM SIGMOD International Conference on Management of Data, pp.99–108, ACM SIGMOD, June 1988.

[2] S. Ghandeharizadeh and D.J. DeWitt, "Hybrid-range partitioning strategy: A new declustering strategy for multiprocessor database machines," Proc. 16th International Conference on Very Large Data Bases (VLDB '90), pp.481–492, Aug. 1990.

[3] H. Yokota, Y. Kanemasa, and J. Miyazaki, "Fat-Btree: An update-conscious parallel directory structure," Proc. 15th International Conference on Data Engineering (ICDE '99), pp.448–457, IEEE Computer Society, March 1999.

[4] J. Miyazaki and H. Yokota, "Concurrency control and performance evaluation of parallel B-tree structures," IEICE Trans. Inf. & Syst., vol.E85-D, no.8, pp.1269–1283, Aug. 2002.

[5] R. Bayer and M. Schkolnick, "Concurrency of operations on B-trees," Acta Informatica, vol.9, no.1, pp.1–21, March 1977.

[6] C. Mohan and F. Levine, "ARIES/IM: An efficient and high concurrency index management method using write-ahead logging," Proc. 1992 ACM SIGMOD International Conference on Management of Data, pp.371–381, ACM SIGMOD, June 1992.

[7] H. Yokota, "Autonomous disks for advanced database applications," Proc. 1999 International Symposium on Database Applications in Non-Traditional Environments (DANTE '99), pp.435–442, IEEE Computer Society, Nov. 1999.

[8] J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann, San Francisco, California, USA, 1992.

[9] V. Srinivasan and M.J. Carey, "Performance of B-tree concurrency control algorithms," Proc. 1991 ACM SIGMOD International Conference on Management of Data, pp.416–425, ACM SIGMOD, May 1991.

[10] P.A. Bernstein, V. Hadzilacos, and N. Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesley, Cambridge, Massachusetts, USA, 1987.
[11] D. Ito and H. Yokota, "Automatic reconfiguration of an autonomous disk cluster," Proc. 2001 Pacific Rim International Symposium on Dependable Computing (PRDC2001), pp.169–172, IEEE Computer Society, Dec. 2001.
[12] H. Yokota and R. Abe, "Secondary storage configuration for advanced data engineering," in Nontraditional Database Systems, ch. 14, pp.212–230, Taylor & Francis, London, UK and New York, USA, 2002.
[13] Y. Mond and Y. Raz, "Concurrency control in B+-trees databases using preparatory operations," Proc. 11th International Conference on Very Large Data Bases (VLDB '85), pp.331–334, Aug. 1985.
[14] D. Amarmend, M. Aritsugi, and Y. Kanamori, "PO-P: A concurrency control protocol for parallel B-trees," IPSJ Trans. Databases, vol.45, no.SIG 14(TOD 24), pp.30–38, Dec. 2004.
[15] P.L. Lehman and S.B. Yao, "Efficient locking for concurrent operations on B-trees," ACM Trans. Database Syst., vol.6, no.4, pp.650–670, Dec. 1981.
[16] V. Lanin and D. Shasha, "A symmetric concurrent B-tree algorithm," Proc. Fall Joint Computer Conference (FJCC '86), pp.380–389, ACM and IEEE, Nov. 1986.

Tomohiro Yoshihara received the B.E. degree from Tokyo Institute of Technology, Tokyo, Japan, in 2005. He is currently a master course student in Tokyo Institute of Technology. He is engaged in research on data engineering. He is a student member of DBSJ.

Dai Kobayashi received the B.E. and M.E. degrees from Tokyo Institute of Technology in 2003 and 2005, respectively. He is currently a Ph.D. student in Tokyo Institute of Technology and a JSPS Research Fellow (DC). He is engaged in research on data engineering and storage systems. He is a student member of DBSJ.

Haruo Yokota received the B.E., M.E., and D.Eng. degrees from Tokyo Institute of Technology in 1980, 1982, and 1991, respectively. He joined Fujitsu Ltd. in 1982, and was a researcher at ICOT for the Japanese 5th Generation Computer Project from 1982 to 1986, and at Fujitsu Laboratories Ltd. from 1986 to 1992. From 1992 to 1998, he was an Associate Professor in Japan Advanced Institute of Science and Technology (JAIST). He is currently a Professor at Global Scientific Information and Computing Center in Tokyo Institute of Technology. His research interests include general research area of data engineering, information storage systems, and dependable computing. He is a member of IPSJ, DBSJ, JSAI, IEEE, IEEE-CS, ACM and ACM-SIGMOD.