Space Management in B-Link Trees

Dibyendu Majumdar
dibyendu@mazumdar.demon.co.uk

Abstract

Space management operations can drastically reduce the concurrency of B-link tree structure modification operations. This paper examines some of the issues and potential remedies.

1 Introduction

The B-link tree algorithms described in [IBRA-05] require that during certain structure modification operations (SMOs), the space map page be latched exclusively for the duration of the SMO. This is required for the Split, Merge, Increase-tree-height and Decrease-tree-height operations. It is assumed that there is a single space map page M that contains a bit vector describing which pages are allocated and which are unallocated.

The problem with latching the space map page exclusively during SMOs is that it effectively serializes all SMOs. If, as the paper describes, only one space map page is used, then the effect is global. However, if we assume a design where space allocation data is maintained in a number of space map pages, each containing a bit vector for a range of pages, then the impact is restricted to the range covered by the space map page in question. Even with this concession, the impact on concurrency is likely to be severe, as a single space map page of 8K size can hold allocation data for upto 64k pages.

It is desirable therefore to reduce the time for which the space map page is exclusively latched.

Operation	**Number of pages latched exclusively**	**Space Map Page latched**
Split	2	Yes
Link	1	No
Unlink	1	No
Merge	2	Yes
Redistribute	2	No
Increase-tree-height	2	Yes
Decrease-tree-height	2	Yes

2 Using Nested Top Actions

To avoid having to latch the space map page exclusively at the same time as other pages, one option is to break down each SMO into two steps. In case of SMOs involving page allocation, the page allocation step is logged first, followed by the SMO itself. In case of SMOs involving page deallocation, the SMO is logged first, followed by the page deallocation. To make the action atomic, these need to be logged as redo-undo records, followed by a Dummy Compensation log records [MOHA-92].

The problem with using the nested top action approach is that it negates the benefits of using single redo-only log records. Each operation requires a redo-undo log record, as it must be possible to undo the SMO should it not complete successfully.

3 Separate (de)allocation steps

Another option is to separate the page allocation and deallocation steps into discrete atomic actions. This improves concurrency while retaining the simplicity of the original design, i.e., it allows SMOs to be logged using single redo-only log records. However, it introduces the problem that if the system fails between the page allocation and the SMO, or between the SMO and the page deallocation, then the space map information may incorrectly show some pages as allocated even though they are not part of the B-link tree. It is worth noting here that the discrepancy in the space allocation map does not impact the correctness of the B-link tree structure, it merely leads to some wasted space. As such, this may be an acceptable solution in many cases.

4 Proposed solution

Ideally we would like a solution that minimizes the duration and frequency of latching space map pages, and yet supports the simplicity of redo-only logging of SMOs. We would also like to avoid the situation where pages end up showing allocated in the space map but are in reality not part of the B-link tree.

The design of the solution is based upon following observations.

1. In ARIES, a Compensation log record is redo-only, and is linked via the undoNextLsn to the predecessor of the undoable log record for which the CLR is compensating [MOHA-92]. A Dummy CLR is a special case, as it does not actually compensate for an undo, instead it is used as a linking mechanism to bypass the undo of actions that are part of the nested top action. We note that there is no reason why an ordinary redo-only log record may not also be used as a Dummy CLR. The only difference would be that such a CLR would contain redo information, whereas a Dummy CLR does not contain such information.

2. During page allocation and deallocation, it may be necessary to scan the space map pages in order to locate the correct space map page. We note that the scan during page deallocation can be avoided if we make a note of the appropriate space map page during page allocation.

3. During an operation that deallocates a page, the deallocated page is exclusively latched, and is also a part of the redo-only log record. This provides us an opportunity to add extra information within the deallocated page to indicate its new status.

5 SMOs involving page allocation

The solution proposes that SMOs that result in page allocation should be handled as nested top actions with the following modifications:

1. The page allocation step should be logged as redo-undo so that in case the SMO aborts, the page allocation will be undone. This ensures that the page allocation is automatically undone at system restart without need for any special action.

2. The SMO should be logged as a Compensation log record. Its undoNextLsn should be set to the Lsn prior to the one that describes the page allocation step. This ensures that the SMO is logged as redo-only, thus avoiding the overhead of supporting undo operation. However, at the same time, we allow the page allocation to be undone in the event that the SMO is not logged.

3. The identity of the space map page responsible for maintaining the allocation status of the new page, should be stored inside the page as a separate field. This makes it possible to avoid searching for the appropriate space map page during page deallocation.

4. A page allocation flag should be set within the newly allocated page to indicate its allocated status. This flag can be used to mark deallocated pages as well, as we shall see in the next section.

6 SMOs involving page deallocation

The solution proposes that an SMO involving page deallocation should be split into separate discrete atomic actions as described below.

1. The structure modification operation should be logged as a redo-only log record. The deallocated page should be marked as deallocated using the page allocation flag within the page. Also, an action to deallocate the page should be enqueued. The action should specify the page to be deallocated, as well as the space map page that owns allocation data for the page.

2. A background process should be configured to listen on the deallocation queue. When a request for page deallocation arrives, it should generate a redo-only log record for the specified deallocation and then perform the deallocation. This must be committed as an independent transaction. Note that the process may batch together updates to several pages that are managed by the same space map page, rather than handling each update individually.

7 Garbage collection of deallocated pages

If the system fails before all the pages that are in the deallocation queue have been logged, then some pages will show as allocated in the space map pages but will not be part of the B-link tree. These pages can be recovered using a single scan of all the pages in the B-link tree after system recovery. Since deallocated pages have a flag set within the page, the garbage collection process does not need to inspect any other attribute of the page to determine the status of the page. The garbage collection process simply enqueues a page deallocation request, after checking that there is not already a deallocation request for the same page. This check is necessary to avoid conflicts with other concurrent processes that may be deallocating pages.

Note that the garbage collection process only needs to obtain shared latches on the pages while inspecting them. Ideally, the pages inspected by the garbage collection process should not be made resident in the Buffer cache, to avoid filling up the buffer cache with infrequently accessed pages. If they are to be part of the Buffer cache, then they should be placed at the LRU (least recently used) end of the cache.

The garbage collection process may be started anytime after system recovery has been completed. It needs to scan each B-link tree only once. It can run in parallel with other processes as a low priority task.

If the system can cope with some amount of wasted space, then the garbage collection process can be run during off-peak periods, possibly at weekends.

8 Advantages of proposed solution

1. Advantages of logging SMOs as redo-only log records are retained.

2. Page allocations are handled normally as nested top actions, thereby requiring no special action.

3. Deallocations are handled asynchronously. By batching several deallocations together, and by avoiding searches for space map pages, the load on space map pages is reduced.

4. SMOs do not require space map pages to be latched exclusively at the same time as other pages are latched exclusively. This allows multiple SMOs to proceed concurrently.

9 Disadvantages of proposed solution

1. A separate background process is required for handling page deallocation requests. However, this has the benefit that deallocation requests can be batched together to reduce latching of the space map pages.

2. A separate garbage collection process is needed to recover deallocated pages that are incorrectly marked as allocated in the space map page. However, this process can run concurrently with other processes, and needs to only scan a B-link tree once. Also, it does not acquire exclusive latches on any page.

10 Another approach to page deallocation

In the approach described above, page deallocations are handled asynchronously. With the help of additional locking, it is possible to devise a solution that handles page deallocations synchronously using nested top actions in the same way as page allocations are handled.

The problem with page deallocations is that they must occur after the SMO that generates them. This means that if we implement such operations as nested top actions, then we need to make the SMO undoable. We have already said why this is undesirable.

To avoid this, we can generate the log record for the page deallocation before the SMO. However, this would cause a problem because other processes may consider the page available for re-use before the SMO is completed, thus corrupting the B-link tree. The solution to this problem is to obtain an exclusive lock on the deallocated page before logging the page deallocation. This lock must be obtained from the Lock Manager. By locking the deallocated page, we prevent other processes from accessing the deallocated page until the SMO is complete. Once the SMO has been logged, the lock is released. Thus, page deallocation locks are held for a short period only, and do not conflict with transaction level locks.

SMOs involving page allocation must respect the page lock. Once a page has been identified as available, we need to obtain an instant duration lock [MOHA-92] on the page to check whether that page is truly available. There is no need to wait for such a lock, therefore the lock request must be conditional [MOHA-92]. If the lock is not available, we can try another page.

In this approach, page deallocations are handled in the same way as page allocations at the expense of additional locking. This approach also avoids the problem of incorrect information in space map pages.

References

[IBRA-05] Ibrahim Jaluta, Seppo Sippu and Eljas Soisalon-Soininen. Concurrency control and recovery for balanced B-link trees. The VLDB Journal, Volume 14, Issue 2 (April 2005), Pages: 257 - 277, ISSN:1066-8888.

[MOHA-92] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh and P. Schwarz. ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. ACM Transactions on Database Systems, 17(1):94-162, March 1992. Also, Readings in Database Systems, Third Edition, 1998. Morgan Kaufmann Publishers.

[GRAY-93] Jim Gray and Andreas Reuter. Chapter 9: Log Manager. Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers, 1993