The ADABAS Buffer Pool Manager

Harald Schöning
Software AG, Uhlandstr. 12, 64297 Darmstadt
hsg@software-ag.de

Abstract

The buffer pool manager is a central component of ADABAS, a high performance scaleable database system for OLTP processing. High efficiency and scalability of the buffer pool manager is mandatory for ADABAS on all supported platforms. In order to allow a maximum of parallelism without facing the danger of deadlocks, a multi-version locking method is used. Partitioning of central data structures is another key to performance. Variable page sizes allow for flexible tuning, but make the buffer pool logic more sophisticated, in particular concerning parallelism.

1 Introduction

SOFTWARE AG's ADABAS is a database system with a very long history. It is successfully used in business-critical OLTP applications which depend on the robustness and the high performance of the underlying database system, such as flight reservation systems, emergency management, etc. Over the years it has been ported to all major operating systems. Today, ADABAS databases can operate on UNIX machines running various flavors of UNIX, WINDOWS PCs, various mainframe operating systems such as MVS, VSE, and BS2000, and other platforms.

To cope with the evolving world of environments, continuos re-engineering of ADABAS has been necessary. In particular, multi-processing architectures (symmetric multi-processing) have caused considerable changes to the whole system. One component which has recently been completely redesigned during this process is the ADABAS buffer pool manager.

The primary task of the buffer pool manager is to cache database pages which have been read from disk, in order to save I/Os if those pages are re-referenced. Changes of the database pages are performed in the buffer pool only and not immediately written to disk. Of course, adequate logging is needed to guarantee the persistence of committed transactions, but logging algorithms are beyond the scope of this paper.

The buffer pool also handles temporary information which is written to disk only if there is a lack of space.

In the following, we will shortly mention some characteristics of ADABAS which are important for the buffer pool design.

1.1 Container Types

ADABAS data on disk are organized in two so-called container files. The DATA container stores the mere data of a database in a compressed form, while the ASSO container stores the schema information, the database translation table (called *Address Converter* in ADABAS), and the indexes. Each record in the DATA container has a unique identifier. The Address Converter maps the unique ID to a physical location. The indexes contain logical identifiers only. There is a third container file called WORK which is used to store temporary data and log information.

Container files can be distributed over an arbitrary number of disks, using raw device or file system access (or mixing both). A container consists of pages, which are numbered in ascending sequence. These pages are the unit of input and output. The buffer pool stores pages from each of the three container types.

1.2 Varying page sizes

The pages within a container may have different size. Depending on the size of the data of a database table and the typical access pattern, the database administrator (DBA) can adjust the page size (in a range of 1 to 32 Kilobytes). For example, if a table is typically accessed with exact match queries via an index, the DBA might choose a page size for the data storage such that a page contains only a few data records. The page size for the

Proceedings of the 24th VLDB Conference
New York, USA, 1998

index, and for the address converter, can be chosen independently. This flexibility in page size can also be used for an adaptation of the database to changing storage medium characteristics, as pointed out in [GG97].

As a consequence, the ADABAS buffer pool manager must cope with pages of various sizes from all three container types. In particular, the varying page sizes affect the buffer replacement algorithms.

Other commercial database systems do not have this freedom in configuration, and, as a consequence, do not need to handle such complex replacement problems. The research database system PRIMA [HMMS87] developed at the University of Kaiserslautern, also supports multiple page sizes. The replacement algorithm implemented there [Si88], however, differs from the one used in ADABAS. There, a *free list* is kept separate from the LRU chain. This list is then copied and a the buffers contained in the LRU chain are consecutively marked as replaceable. When an area has been found which is large enough for the new buffer, the process stops. This algorithm, however, lacks the flexibility of the ADABAS algorithm described below.

1.3 Parallel access

The buffer pool manager has to care for a proper synchronization of the accesses to the pages in the buffer pool. Several threads which execute in parallel on multiple CPUs may want to access the same database page, and hence the same buffer. Of course, the synchronization has to be very efficient, not only avoiding deadlocks, but also keeping waiting times as short as possible.

The following sections discuss the architecture and the algorithms chosen for the new ADABAS buffer pool manager.

2 Architecture of the buffer pool manager

The buffer pool is allocated as a contiguous piece of memory. In order to avoid double page faults [EH84], i.e. page faults in the operating system's virtual memory, the whole buffer pool can be pinned in the physical memory. When a block from disk is read into the buffer pool, a header structure is assigned to it, which stores all information needed for the management of the block, including, of course, the identification of the database pages this block corresponds to. These headers themselves are allocated in the buffer pool in contiguous areas. Note that the variable page size in ADABAS makes it impossible to predict the number of headers needed (of course, the least possible page size determines an upper limit to the number of headers, but allocating that much headers in advance could waste a lot of space).

ADABAS directly references the pages in the buffer pool. Therefore, the address of a page must not change and the page must not be removed from the buffer pool while a command is still working on it. To guarantee this, database management systems usually FIX and UNFIX the pages in a buffer pool explicitly [EH84]. In the ADABAS buffer pool manager, this functionality is combined with the synchronization of page accesses by the ADABAS commands. For this purpose, each header contains a readers/writer lock.

The headers are linked in physical sequence (so-called *physical chain*) and in LRU sequence (*LRU chain*). Furthermore, to enable an efficient search for a specific database page, a hash structure is allocated. Each hash bucket contains the pointers to the corresponding headers and is protected by its own latch. Hence, lock conflicts on the hash structure are rare. The overflow of a hash bucket is organized as AVL tree. Thus, even in the case where a bucket has a large amount of overflow information the access remains fast - another pre-requisite for low lock contention on the hash structure. The buffer pool architecture is depicted in Figure 1 (LRU chain not shown).

3 Page Access Synchronization

The access to a database page works as follows: The page is searched in the hash structure. The hash bucket is protected by a latch (a short time mutual exclusion lock). If it is not found, a header for the database page is allocated, exclusively locked, and entered into the hash structure such that other tasks have a reference to it. Then the physical I/O is started. When it is finished, the exclusive lock can be downgraded to a shared lock if the database pages was needed for reading only.

If the page had been found in the hash structure, the buffer pool tries to acquire a lock of the requested quality (shared or exclusive) and, if successful, returns a pointer to the corresponding location in the buffer pool.

Locks on database pages in the database are held until the command has performed its changes to the page, i.e. for a very short time only.

When database pages are logically linked (e.g. nodes in an index tree), and updates affecting this link have to be performed, more than one page has to be exclusively locked at a time. While deadlocks in such situations often can be prevented by enforcing a certain sequence of locking, this is not possible in all cases. For example, positioning in an index is done from root to leave, while index updates occur from the leave to the root. Hence, positioning and updating simultaneously could lead to a deadlock. Locking the whole index would lead to unacceptable waiting situations. To cope with this situation, the ADABAS buffer pool manager uses a multi-version locking scheme: when an exclusive lock is not granted, the buffer pool tries to acquire a shared lock. If this is granted, a copy of the database page is generated in

the buffer pool, and the pointer in the hash structure is set to this copy. Hence, all threads that subsequently search for this page will find the copy. The block containing the original page remains in the buffer pool, but is placed at the end of the LRU chain, thus being prime candidate for replacement once all (shared) locks on it are released. Its

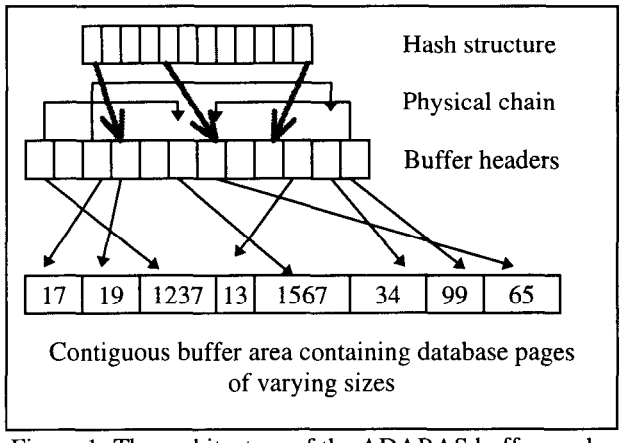


Figure 1: The architecture of the ADABAS buffer pool

access time stamp is set to zero.

A consequence of this locking scheme is that shared locks do not protect a logical link between pages against changes. Therefore, pages found by following a link always have to be re-evaluated before they can be used. Example 1 illustrates this effect.

Note that in high-load situations, more than one thread could copy the same database page. Only one of these two copies must survive. For this purpose, the exchange in the hash structure for search must be atomic. It fails if the address to be replaced is not the expected one.

4 Saving Changes to Disk

The ADABAS buffer pool managers saves changed pages to disk in an asynchronous manner. When a certain (configurable) percentage of all pages has been modified, an asynchronous thread (called the *buffer flush thread*) starts to write all changed pages to the disk. Of course, the pages must not be modified while they are written to disk. On the other hand, it is not acceptable to defer changes to those pages until the writing has been done. If a page which is locked by the buffer flush thread is to be changed by another thread, the same multi-version locking as described above is applied. The pages involved in the buffer flush are locked by the buffer flush thread using a privileged read lock. If a page is currently write locked, it is entered into a *refused-lock list* and skipped. After all other pages are locked, the buffer flush thread blocks on the pages in the refused-lock list if necessary. Typically, the updating command that had held a lock on those pages has meanwhile released the lock

The asynchronous writing of changed pages has some consequences:

- A page which has been chosen for replacement need not be written to disk before it is replaced, allowing a fast replacement.
- Pages cannot be replaced if they had been changed after the last buffer flush. Therefore, a buffer flush must occur before too many pages are "dirty". On the other hand, flushing too early destroys the caching effect for updates because pages are written to disk after fewer updates per page. Obviously, finding a reasonable percentage of dirty pages for the start of a buffer flush is not trivial. In order to relieve the DBA from this task, ADABAS can choose a useful percentage and internally adapt it to the current situation.
- The crash recovery algorithms are tightly coupled to the asynchronous writing. The start and the end of the buffer flush are logged. From this logging information the crash recovery algorithm can infer which database changes are already reflected on disk and which are (possibly) not. Therefore it is essential that all changed pages are covered by the buffer flush. On the other hand, pages which are very frequently updated could defer the whole buffer flush considerably. To cope with such situations, the buffer flush can be split into a first part which contains all pages which could be locked without blocking on the lock, and a second part which flushed all the other pages (and usually handles very few pages).

5 Buffer Replacement Handling

As pointed out earlier, buffer replacement in ADABAS is quite sophisticated. If a page of a certain page size is to be read into the buffer pool, and the buffer pool is filled up (which is the normal case after an initial filling phase), the necessary space must be provided by selecting another buffer which can be overwritten. However, caused by the varying page sizes in an ADABAS database, it might be necessary to overwrite several other pages of smaller page size. In databases with one fixed page size, the first available page when searching from the end of the LRU chain can be chosen for replacement. This is not true for ADABAS.

Consider the following case: A page with size 4 KB has to be read into the buffer pool. At the end of the LRU chain, only 2 KB pages can be found. The next 4 KB page is quite at the begin of the LRU chain, i.e., it is a quite new page. In this case, one of the 2KB pages and its physical neighbor should be replaced. However, the physical neighbor might also be a very new page. To find good replacement candidates, the following handling is applied. The LRU chain is searched from its end. When a page is found which is available for replacement (i.e. contains no unwritten changes and is not locked), ADABAS searches for the necessary space starting from this page. The left neighbors are considered, as long as

they can be replaced and the necessary space is not yet gathered. Then, the right neighbors are checked. The space between the left-most neighbor found and the right-most one usually leaves several choices for so-called *overlay sets*, i.e. sets of buffers which could be replaced to gain the needed space (cf. Figure 2). The overlay set with the lowest costs is stored.

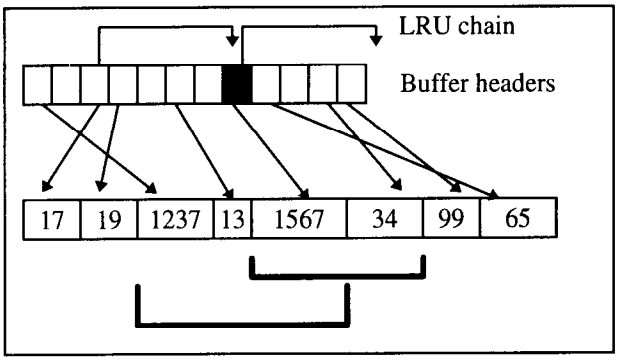


Figure 2: Two overlay sets based on page 1567

Then the LRU search is continued, until an upper limit of found pages is reached, or until a single page is found which is large to render the necessary space. This page is a singleton overlay set. The cheapest overlay set is chosen for replacement.

The cost of an overlay set is determined by applying a function to the access time stamps of the pages in the set. The choice of this function heavily influences the replacement algorithm. If the function is MIN, for example, the first overlay set found would be selected. If the function is SUM, the likelihood of a multi-buffer replacement decreases rapidly with the number of pages. In the case of MAX, an overlay set is chosen only if all its pages are older than the oldest replaceable single page of sufficient size.

Obviously, search for replacement candidates is an operation which takes quite long due to the need to cope with different page sizes. Unfortunately, the LRU chain cannot be changed while such a search is performed. Since every access to a database page should update the LRU chain (placing the accessed page in front of the LRU chain), there is a considerable bottle neck. To avoid lock contention on the LRU chain, ADABAS splits the buffer pool into several physical regions, where each region has its own LRU chain. The number of regions depends on the size of the buffer pool, the maximum parallelism allowed by the DBA, and other criteria.

These physical regions are chosen for replacement in a round-robin manner. Only the affected LRU chain is locked. Furthermore, the updates to the LRU chain are deferred. Obviously, the updates need not be done before a replacement search is performed. Hence, the access to pages is memorized, but it is reflected in the LRU chain only when the lock for replacement search is required anyway. Note, that in contrast to the procedure used in ORACLE [Br97], the ADABAS algorithms reflects the correct sequence of accesses in the LRU chain.

6 Prefetching

Sequential operations which scan the data of multiple adjacent database pages are quite common. In ADABAS, the DBA can optimize for such operations, e.g. by choosing large page sizes. However, such optimizations are complex and might decrease the performance of commands with different access patterns. Therefore, ADABAS recognizes sequential access, and can read several pages in one IO into a contiguous buffer pool area. The replacement algorithm described above obviously covers this case without adaptation. Although read with one I/O, all pages have their own header and are managed by the buffer pool manager as if they has been read separately. The number of pages to be read in one IO is dynamically determined according to the following criteria:

- Maximum number of pages needed by the current command
- Number of pages which fit into a single physical IO. This is platform dependent, but it also depends on the distribution of the container files over disks.
- Next page which is already in the buffer pool. In order to avoid inconsistencies with updates on this page, the page must not be re-read into the buffer pool.
- Available space in the buffer pool

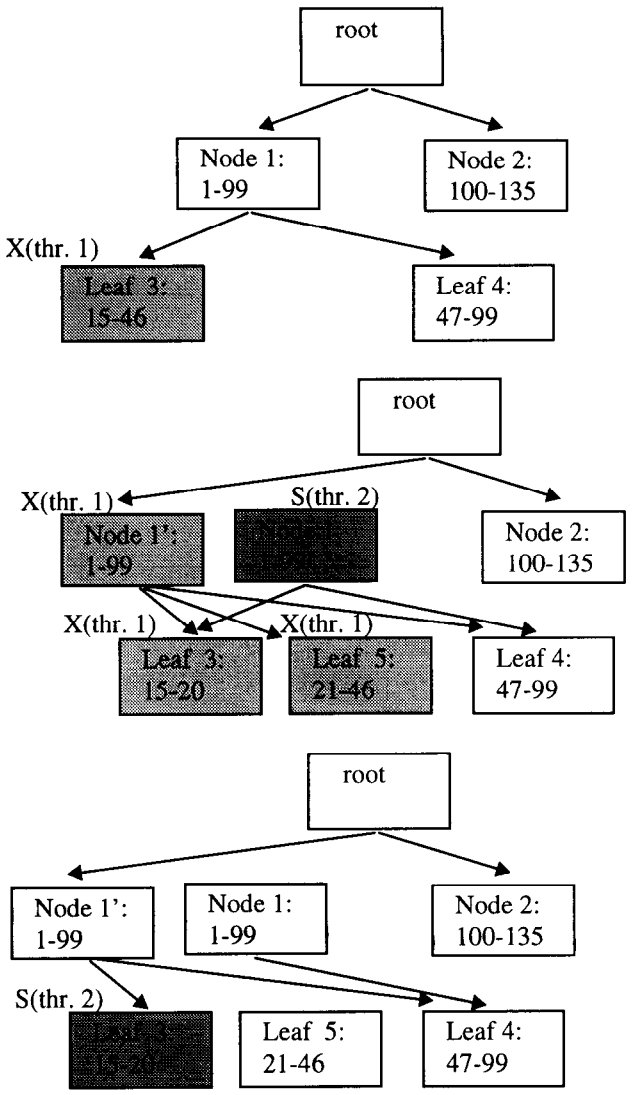
7 Summary

The ADABAS buffer pool manager has completely been redesigned for the latest parallel version of ADABAS. In particular, the locking of the LRU chain had been a bottleneck, in particular because the replacement algorithm is very complex due to the different page sizes used in a database. To prevent lock contention on the LRU chain , the chain has been split into several areas. Furthermore, the update of the LRU chain is done lazily. In order to increase parallelism and avoid deadlocks in particular in index operations, dedicated multi-version locking protocols have been introduced. Various dynamic optimizations relieve the DBA from too sophisticated tuning. The use of further self-tuning algorithms such as LRU-2 [OOW93] is constantly investigated.

a) Thread 1 wants to insert value 20 into the index. It locks leaf 3 exclusively.

b) Thread 2 wants to read value 30. It acquires a shared lock on node 1. Before it gives up the lock again, thread 1 tries to lock node 1 because leaf 3 has to be split due to lack of space for value 20. It does not get the lock and creates a copy of node 1. Then it creates a new leaf 5.

c) Thread 2 had blocked on the shared lock on leaf 3. After thread 1 has finished thread 2 gets the shared lock now. The value 30, however, cannot be found in leaf 3 any longer. Thread 2 must check whether the version of node 1 that it had seen is still the current one. If so, the value 30 is not in the index, otherwise thread 2 must repeat the positioning.



Example 1: The need for repositioning

References

Br97 Bridge, W., et al: The Oracle Universal Server Buffer Manager, in: Proc. 23rd Int. Conference On Very Large Data Bases, VLDB 97, pp. 590-594.

EH84 Effelsberg, W., Härder, T.: ACM Transactions on Database Systems, Vol. 9, No. 4, Dec. 1984, pp. 560-595.

GG97 Gray, J., Graefe, G.: The Five-Minute Rule Ten Years Later, and Other Computer Storage Rules of Thumb, in: SIGMOD RECORD Vol. 26, No. 4, Dec. 1997, pp. 63-68.

HMMS87 Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA - A DBMS Prototype Supporting Engineering Applications, in: Proc. 13th VLDB, 1987, pp. 433-442.

OOW93 O'Neil, E.J., O'Neil, P. E. , Weikum, G.:The LRU-K Page Replacement Algorithm for database disk buffering, in: Proc. ACM SIGMOD Int. Conf. On Management of Data, 1993, pp. 297-306.

Si88 Sikeler, A.: VAR-PAGE-LRU: A Buffer Replacement Algorithm Supporting Different Pages Sizes, in: Proc. Int. Conf. On Extending Database Technology, EDBT88, Venice, Italy, Springer-Verlag, Berlin, 1988, pp. 336-351.