

Buffer Management

Justin Falk
Robert Morris College
925 W Huron St Apt 215
Chicago, IL 60622
001-219-614-4265
jrfalk@gmail.com

ABSTRACT

In this paper, many details of database buffer management including goals, roles, organization, and schemes are discussed. There is a focus on the LRU-K scheme and its features.

Categories and Subject Descriptors

H.2. [Database Management]: Buffer management of DMBS – *algorithms, schemes, roles.*

General Terms

Buffer, Management, Algorithms, Schemes, Performance, Memory, Classification, Physical Storage

Keywords

Buffer, Page, Page Faults, Hit, Hit Rate, CLOCK, FIFO, LFU, LRU, LRU-K, DBMIN, Locality Set, Locality of Reference, Hot Set, Hot Points, Latching

INTRODUCTION

The following is a brief explanation of database buffer management. Buffer Management is explained using generic terms common to all types of databases. The buffer organization and its goals are discussed. Many types of schemes are used to accomplish an ideal buffer management system.

1. BUFFER MANAGEMENT

A buffer management system must be in place for a relational DBMS to quickly recall and write data to memory for use. The buffer acts as a liaison between the physical storage and the database system. The buffer manager is the component that controls this buffer and lends its use to other components of the database that need its information. The buffer management system should be setup to provide these other components with this information in the speediest manner possible.

2. ORGANIZATION OF THE BUFFER

The buffer is composed of multiple frames or control blocks. Each frame has a specific memory capacity. The number of frames can be changed for optimal configuration but are static while the database is running. Frames are ideally setup to match the size of a page. A page is a section of the physical database storage area. When information is requested the buffer frame will be filled up with data it retrieved from the page. These frames within the buffer are fast volatile memory units most advantageous for feeding the data to the components that

requested them. All frames together make up a unit called the Buffer Control Block Table. Each block in the table contains addresses, counters, id keys, bits, and other mechanisms used in the buffer management scheme.

3. ROLES OF THE BUFFER

The buffer has many purposes [1]. As mentioned previously, its primary focus is to share data. If multiple requests are sent for the same data the buffer will provide this data repeatedly rather than have to access and read the data numerous times from the physical storage pages. Another role of the buffer is its storage durability. Log files need to be created and the buffer manager will use its storage memory to write information to the log files. Synchronization is another role performed by the buffer manager. Every time another component requests data from the buffer manager a process called “latching” results. This latching is the synchronization or obtaining of the page information with the component requesting to access the page. Besides serving other components that need data, the buffer also performs the role of writing data back to the physical pages. The buffer will retain this information in its frames after an update or new data gets created and then asynchronously write the data in the page. Asynchronously refers to the task being completed not related to or dependent on the information or any other task being performed. This allows for the buffer to write to the page after the fact at a time that is more convenient as not to stress other operations of the database.

4. BUFFER MANAGEMENT GOALS

The primary goal of the buffer manager is to minimize the number of page faults and maximize the hit rate [1]. A page fault happens when information requested is not located in the buffer and the request has to go to the physical storage. A hit is when the information requested is found in the buffer. The hit rate ratio is the amount of hits compared to the amount of hits found in a period of time. The hit rate ratio should be close to 100% as possible. The closer to 100% the ratio the more time is saved by accessing the data from the buffer compared to the accessing the data from the physical storage which takes longer. Although the time may only be tenths or hundredths of a second this done repetitively over again adds up. Also other components then have to wait on this information causing a chain effect up the component ladder. To obtain the highest hit ratio the buffer clearly needs to keep pages that are most frequently accessed in its frames. Predicting which future pages will be requested is done through observation of pages requested in the near past [1]. This is called locality of active data. As not to fill up the buffer

with unnecessary page data, information that will probably not be requested compared to information that probably will be requested will not be given precedence within the frame. This is called locality of passive data. Both of these localities are referred to as localities of reference, meaning which pages referenced to likelihood of lookup should be kept in the buffer.

5. BUFFER SCHEMES

There are many types of schemes used by the buffer manager based on the type of database implemented. Each has its own unique capability of maintaining requested information in the buffer based on the type of relational system needed.

5.1 Buffer Scheme Classification

Buffer Management Schemes usually fall into three categories [1]. Global schemes consist of a universal buffer that is not divided or broken into different sections or partitions. Each request or transaction to information on the same page is treated equally and a shared copy of the buffer is provided. Page-type oriented schemes splits the buffer space up into multiple segments called partitions. Each partition is made up of a common type of data such as system data, counters, page indexes, etc. Transaction-oriented schemes distribute frames within the buffer based upon new requested information from the query or transaction requested. Each request uses its own divided segment of frames decided dynamically or statically.

5.2 Common Global Schemes

FIFO (First In-First Out) uses chronological age as its only variable. When the buffer is completely full the oldest frame in the buffer is replaced with the new page from physical storage. A single pointer indicates which frame is the oldest. Every time the oldest frame is replaced the pointer jumps to the next oldest frame indicating it should be replaced next. The disadvantage of this occurs when the same information is requested repeatedly and the buffer pages have to be replaced unnecessarily. CLOCK is a slightly modified version of FIFO which contains a reference bit (a 1 or 0) within each frame of the buffer. If a frame was used in the most recent transaction or query the bit contains a 1, otherwise it contains a 0. The pointer will skip over any frame with a 1 therefore not replacing it. Only the oldest frame which did not contain any requested information gets replaced. If all bits contain 1 then the oldest frame gets replaced. The LFU (Least Frequently Used) scheme contains a reference bit similar to a counter that contains a numerical value representing the last time the frame was used. Every time the frame is used the bit gets incremented by 1. If the next transaction does not use the frame the number gets reset back to 1. Frames constantly referenced will have high counts and not be replaced. Frames with the low counts will be replaced first in order from lowest to highest count. The disadvantage of this is if a very frequently used frame does not get used in the last transaction it will most likely be replaced. Because of this phenomenon this scheme is infrequently used. The LRU (Least Recently Used) scheme is a frequently used scheme that is employed in one form or another in most commercially used databases today [1]. The frame that has not been accessed in the longest amount of time gets replaced with the new page information. Frames are kept in a chain order with the least recently used page on top and the most recently used page at the bottom. The top frames will be replaced first.

Although this scheme is very useful it does have its disadvantages and can be improved upon. Its overhead, the ability to keep the frames in order, is a bit intensive on memory. Also this scheme has no way to keep track of which frames are frequently accessed compared to which frames are infrequently accessed besides knowing the last time it was used.

5.2.1 Least Recently Used -K Scheme

LRU-K was designed to address some of the flaws of LRU. The K value represents how many previous transactions the buffer manager will keep track of. This is a value greater than or equal to 2. When a frame has to be replaced in a buffer the frame with the smallest time with a value lower than the chosen K value is replaced. The K value is unique in that the buffer keeps track of all frames that have referenced a page greater than or equal to its K value. All frames greater than this K value are excluded from replacement and only frames lower than K value are considered when looping through the frames. This requires smaller overhead than simple LRU scheme and can easily determine which frames have longer time intervals between requested transactions but are still accessed frequently. This creates an adaptive process which can distinguish the "locality of reference" between pages in the physical storage across numerous transactions over a chosen time. Because of the design of LRU-K pages that have made it into a buffer frame can be distinguished to not be immediately replaced by using a value of time (usually n seconds) called a correlated reference period. Only after this correlated reference period has surpassed can this particular frame be replaced. This greatly helps to eliminate an early frame replacement problem in which frequently used pages do not stay in the frames because they are flushed out by a large amount of transaction references or bursting of transaction volume. Another development of LRU-K is the addition of a history block to help prevent frequently referenced pages that can become flushed out of the buffer due its lack of timed frequency between transactions. A history block is written every few minutes to system memory outside of the buffer, and the buffer can be used to recall this information from the system memory at a significantly quicker rate than it could from physical storage pages. LRU-K scheme also has two data structures used to keep track of most recent used references. The first is a HIST(P) structure that contains the number of times a page, P, is referenced up to K times. The second is a LAST(P) structure that stores the time of the most recent reference. Complex algorithms for hits and misses are used for the formulas of HIST(P) and LAST(P). Experiments [1] have shown that a K value of 3 (LRU-3) has approximately the same hit ratio as a K value of 2 (LRU-2), but has significantly less success when there are frequent changes in transaction request patterns. LRU-2 outperforms LRU in nearly all experiments when buffer sizes contain 1000 or fewer frames. However no significant difference is noticed for large buffers containing 2000 or more frames.

5.3 Common Non-Global Schemes

The "Hot Set" algorithm uses advanced knowledge of reference patterns to find a "Hot Set" of pages. These pages are frequently used many times over in succession or in a series. This algorithm uses a looping model to plot page faults and buffer size to find "hot points" [2]. With these "hot points" mapped the set is then examined in separate buffers, called domain buffers, using the LRU scheme. This management scheme can often lead to over-

allocation of memory. Another scheme is called DS (Domain Separation) algorithm. Each page that makes it into the buffer is classified into a type. These types are then sorted through individually in a separate group of domain buffers using the LRU scheme. The problem with this type of scheme is the types are not weighted or considered, and memory allocation must be static. Another scheme is called the “New” algorithm where the buffer pool is divided and memory is allocated on a relationship basis. Each subdivided set is given a rank in a priority list. The Most Recently Used scheme is used inside each set. The priority list is decided based upon changing heuristics, which are found out through discovery of hits [3]. The problems with the “New” scheme is that searching through each subdivided set using MRU can be time consuming and much memory must be allocated to perform this. Another management scheme is called DBMIN, and it uses many buffers to manage frames on an individual query basis to create a file instance. This divided set is associated with a “locality set” [3]. Queries have operations or patterns that are very predictable. This is called a “locality set” and the file instance has ownership over the locality sets. If an operation does not belong to a locality set it goes in a global buffer where all erroneous operations are contained. All the divided buffers are accessible through others through a global buffer table. The algorithm allows for three scenarios for replacement. The advantage of this algorithm is many scenarios can be stored and many of the operations can quickly be provided based on the query type. Another wards, it is a smart system that knows where to look for its data similar to a very well-documented and indexed storage system loaded in memory. A disadvantage of DBMIN is the time constraints created when multiple users create numerous file instances causing long wait times [1]. Another problem is that all file instances are looked at independently and there is no memory sharing outside of each instance.

6. BUFFER MEMORY

Databases can be configured in a shared or dedicated architecture. The choice can highly impact the performance of the buffer [4]. If the buffer is too small then the hit ratio will become very low. If the buffer is too large then many pages will be accessed to fill all of the frames creating a paging problem [5]. Many values can be set in the parameters of a dedicated environment. Each parameter is unique in that a value too small may hinder or

restrict system performance. However, a value too large may overload the capacity if multiple users or transactions are taking place. In a shared environment dynamic memory must be balanced between the shared pool of the buffer and the System Global Area memory. Too much or too less in either area can lower overall system performance.

7. ACKNOWLEDGMENTS

My thanks to the many free educational sites that believe in sharing valued information. Specialized database information is often secured or special membership is needed to access digital libraries. Much appreciation is also given to many professors of college universities who publish information to public websites for database courses.

8. REFERENCES

- [1] Illinois Institute of Technology faculty, *Buffer Management*, <http://www.cs.iit.edu/~cs525/slides/chapter3.pdf#search=%22role%20of%20buffer%20manager%22>, CS525 faculty, 2006
- [2] Chou, H., and DeWitt, D. J *An Evaluation of Buffer Management Strategies for Relational Database Systems*, Morgan Kaufmann Publishers Inc, 1985
- [3] Gribble, S., and Fox A., Summary of *An Evaluation of Buffer Management Strategies for Relational Database Systems*, Date unknown
- [4] Noriega, A., *A Robust Blueprint to Performance Optimization in Multiblock Databases*, <http://www.nyoug.org/Presentations/2004/200409mblk.pdf#search=%22shared%20vs%20dedicated%20database%20buffer%22>, 2004
- [5] Shasha, D., and Bonnet, P., *Database Tuning Principles, Experiments and Troubleshooting Techniques*, <http://www.distlab.dk/dbtune/slides/dbtune.pdf#search=%22OS%20hardware%20and%20tuning%20shasha%20philippe%20%22>, 2002