TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Department of Computer Science
Database and Information Systems Group

Project Thesis:

Performance Evaluation of Different Open Source and Proprietary Implementations of Data Structures in the context of a DBMS Buffer Manager

by **Max Fabian Gilbert***

Day of release: January 31, 2019

*m_gilbert13@cs.uni-kl.de

Abstract

Needless to say, every database management system needs to be able to manage data. The data structures used to manage those data in a database have a major influence on various characteristics (e.g. performance) of a database management system and therefore, the usage of specific data structures (e.g. B-tree indexes) and even some implementation details of those are very important decision in DBMS design.

But for correct and performant operation, a DBMS needs to manage various kinds of meta data as well. Some of those meta data needs to be persistent (e.g. the catalog of a relational DBMS) but some can also be non-persistent. Because of the non-persistence of data managed by the buffer management of a DB, the meta data required for the buffer manager are also usually non-persistent. The data structures used to manage those meta data are—unlike the data structures used to manage the data—more an implementation than a design decision. For some kinds of those meta data, it's—due to the non-criticality of the specific meta data management—even reasonable to use data structures provided by the used programming language even though there might be more performant data structures for the purpose. But more performant implementations for most of those data structures don't need to be implemented specifically for one project, there are many different implementations available in open source and proprietary libraries.

This work is a performance evaluation of various MPMC

This page intentionally left blank.

Contents

This page intentionally left blank.

1 Buffer Frame Free List

1.1 Purpose

A buffer manager is required for every disk-based DBMS. A disk-based DBMS stores the pages of a database on secondary storage but to read and write pages, they are required to be in memory.

This feature is provided by the buffer pool management by managing the currently used subset of the database pages in buffer frames located in memory. A buffer frame is a portion of memory that can hold one database page and each of those frames got a frame index as identifier.

During operation, database pages are dynamically fetched from the database into buffer frames. Once a page is not required anymore, it might be evicted from the buffer pool freeing a buffer frame.

Due to the fact that pages are only allowed to be fetched into free buffer frames, the buffer manager needs to know all the free buffer frames. Therefore, a free list for the buffer frames is required.

1.2 Compared Queue Implementations

To ease implementation of page eviction strategies like CLOCK, a free list should use a FIFO data structure like a queue. Therefore the buffer frame freed first is (re-)used first as well.

Almost every state-of-the-art DBMS supports multithreading and therefore, there are usually multiple threads concurrently fetching pages into the buffer pool and evicting pages from the buffer pool. Following this, a buffer frame free list has to support thread-safe functions to add frame indexes to the free list and to retrieve/remove frame indexes from it. Queues providing those thread-safe access functions are usually called multi-producer (add frame indexes) multi-consumer (retrieve/remove frame indexes) queues.

An approximate number of buffer indexes the free list must also be provided by any free list implementation to support the eviction of pages once there are only a few free buffer frames left. Thread-safe access to this number is desirable but not absolutely required.

1.2.1 Boost Lock-Free Queue with variable size

The famous *Boost C++ Libraries* offer a lock-free MPMC queue in the library `Boost.Lockfree`. Like most other non-blocking data structures, this MPMC queue uses atomic operations instead of locks or mutexes. To support queues of dynamically changing sizes, this queue implementation also uses a free list internally.

The non-thread-safe construction/destruction of the data structure is no limitation for the purpose as a buffer frame free list because the buffer pool of our prototype system is constructed single-threaded. This data structure does not offer the number of contained elements and therefore, an approximate number of buffer indexes the free list needs to be managed outside.

1.2.2 Boost Lock-Free Queue with fixed size

This data structure is identical to the data structure in Subsection 1.2.1 but does not use dynamic memory management internally. Therefore, the capacity of the queue (i.e. the maximum number of buffer frames of the buffer pool) needs to be specified beforehand which allows the usage of a fixed-size array instead of dynamically allocated nodes.

1.2.3 CDS Basket Lock-Free Queue

[HSS07]

1.2.4 CDS Flat-Combining Lock-Free Queue

1.2.5 CDS Michael & Scott Lock-Free Queue

1.2.6 CDS Variation of Michael & Scott Lock-Free Queue

1.2.7 CDS Ladan-Mozes & Shavit Optimistic Queue

1.2.8 CDS Michael & Scott Blocking Queue with Fine-Grained Locking

1.2.9 CDS Segmented Queue

1.2.10 CDS Vyukov's MPMC Bounded Queue

1.3 Performance Evaluation

Bibliography

[HSS07] Moshe Hoffman, Ori Shalev, and Nir Shavit. "The Baskets Queue". In: *Principles of Distributed Systems.* Lecture Notes in Computer Science (4878 2007): *11th International Conference, OPODIS 2007 Guadeloupe, French West Indies, December 17-20, 2007 Proceedings.* Ed. by Eduardo Tovar, Philippas Tsigas, and Hacène Fouchal. Found. by Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen, pp. 401–414. DOI: 10.1007/978-3-540-77096-1_29. URL: https://people.csail.mit.edu/shanir/publications/Baskets%20Queue.pdf (visited on Jan. 24, 2019).