

Memory-Mapped Transactions

by

Jim Sukha

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2005

© Massachusetts Institute of Technology 2005. All rights reserved.

Author
Department of Electrical Engineering and Computer Science
May 19, 2005

Certified by
Charles E. Leiserson
Professor of Computer Science and Engineering
Thesis Supervisor

Certified by
Bradley C. Kuszmaul
Research Scientist
Thesis Supervisor

Accepted by
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Memory-Mapped Transactions

by
Jim Sukha

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Memory-mapped transactions combine the advantages of both memory mapping and transactions to provide a programming interface for concurrently accessing data on disk without explicit I/O or locking operations. This interface enables a programmer to design a complex serial program that accesses only main memory, and with little to no modification, convert the program into correct code with multiple processes that can simultaneously access disk.

I implemented LIBXAC, a prototype for an efficient and portable system supporting memory-mapped transactions. LIBXAC is a C library that supports atomic transactions on memory-mapped files. LIBXAC guarantees that transactions are serializable, and it uses a multiversion concurrency control algorithm to ensure that all transactions, even aborted transactions, always see a consistent view of a memory-mapped file. LIBXAC was tested on Linux, and it is portable because it is written as a user-space library, and because it does not rely on special operating system support for transactions.

With LIBXAC, I was easily able to convert existing serial, memory-mapped implementations of a B⁺-tree and a cache-oblivious B-tree into parallel versions that support concurrent searches and insertions. To test the performance of memory-mapped transactions, I ran several experiments inserting elements with random keys into the LIBXAC B⁺-tree and LIBXAC cache-oblivious B-tree. When a single process performed each insertion as a durable transaction, the LIBXAC search trees ran between 4% slower and 67% faster than the B-tree for Berkeley DB, a high-quality transaction system. Memory-mapped transactions have the potential to greatly simplify the programming of concurrent data structures for databases.

Thesis Supervisor: Charles E. Leiserson
Title: Professor of Computer Science and Engineering

Thesis Supervisor: Bradley C. Kuszmaul
Title: Research Scientist

Acknowledgments

These acknowledgments are presented in a random order:

I would like to thank my advisor, Charles E. Leiserson for his helpful comments on this thesis, and on his helpful advice in general.

I would also like to thank Bradley C. Kuszmaul, who has been deeply involved in this project from day one. Bradley's initial idea started me on this project that eventually became LIBXAC, and I have had many helpful discussions with him on this topic ever since.

I'd like to thank Bradley for giving me an implementation of a B^+ -tree, and Zardosht Kasheff for the code for the cache-oblivious B-tree. The experimental results on search trees without LIBXAC are primarily the work of Bradley, Michael A. Bender, and Martin Farach-Colton.

All of the people in the SuperTech group deserve a special round of acknowledgments. Those people in SuperTech, also in random order, include Gideon, Kunal, Jeremy, Angelina, John, Tim, Yuxiong, Vicky, and Tushara. Everyone has been wonderfully patient in listening to my ramblings about transactions, names for LIBXAC, research, and life in general. They have all kept me (relatively) sane throughout the past year. In particular, I'd like to thank Kunal, Angelina, and Jeremy for their feedback on parts of my thesis draft, and more generally for choosing to serve the same sentence. I'd also like to thank Ian and Elizabeth, who are not in the SuperTech group, but have also been subjected to conversations about transactions and other research topics. Thanks to Leigh Deacon, who kept SuperTech running administratively.

Thanks to Akamai and MIT for the Presidential fellowship that funded my stay here this past year. Also, thanks to the people I met through SMA for their helpful comments on the presentation I gave in Singapore.

Thank you to my family and to my friends, here at MIT and elsewhere. Without their support, none of this would have been possible.

I apologize to all those other people I am sure I have missed that deserve acknowledgments. I will try to include you all when the next thesis rolls around.

This work was partially supported by NSF Grant Numbers 0305606d and 0324974, and by the Singapore MIT Alliance. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 13 |
| 1.1 | Explicit I/O vs. Memory Mapping | 14 |
| 1.2 | Locks vs. Transactions | 17 |
| 1.3 | Concurrent Disk-Based Programs | 21 |
| 1.4 | Thesis Overview | 22 |
| 2 | The Libxac Interface | 25 |
| 2.1 | Programming with LIBXAC | 26 |
| 2.2 | Semantics of Aborted Transactions | 30 |
| 2.3 | Optimizations for LIBXAC | 32 |
| 2.4 | Related Work | 33 |
| 2.5 | Advantages of the LIBXAC Interface | 35 |
| 3 | The Libxac Implementation | 37 |
| 3.1 | Overview | 37 |
| 3.2 | Transactions on a Single Process | 40 |
| 3.3 | Concurrent Transactions | 43 |
| 3.4 | Conclusion | 46 |
| 4 | A Performance Study | 49 |
| 4.1 | The Hardware | 49 |
| 4.2 | Performance of Nondurable Transactions | 50 |
| 4.2.1 | Page-Touch Experiment | 51 |
| 4.2.2 | Page-Touch with an Advisory Function | 51 |
| 4.2.3 | Decomposing the Per-Page Overhead | 53 |
| 4.3 | Durable Transactions | 58 |
| 4.4 | Testing Concurrency in LIBXAC | 59 |
| 5 | Search Trees Using Libxac | 63 |
| 5.1 | Introduction | 63 |
| 5.2 | Serial B ⁺ -trees and CO B-trees | 66 |
| 5.3 | Search Trees Using LIBXAC | 68 |
| 5.4 | Durable Transactions on Search Trees | 68 |
| 5.5 | Summary of Experimental Results | 71 |
| 6 | Conclusion | 73 |
| 6.1 | Ideas for Future Work | 73 |
| 6.2 | LIBXAC and Transactional Memory | 74 |
| A | Restrictions to the Libxac Interface | 79 |
| A.1 | Restrictions to LIBXAC | 79 |
| B | Transaction Recovery in Libxac | 81 |

| | |
|---|-----------|
| C Detailed Experimental Results | 85 |
| C.1 Timer Resolution | 85 |
| C.2 Page-Touch Experiments | 86 |
| C.3 Experiments on Various System Calls | 86 |
| C.4 Durable Transactions | 90 |
| C.5 Concurrency Tests | 92 |
| C.6 Search Trees using LIBXAC | 92 |

List of Figures

| | | |
|-----|--|----|
| 1-1 | Two versions of a simple C program that reads the first 4-byte integer from each of 5 randomly selected pages of a file, computes their sum, and stores this value as the first 4-byte integer on a randomly selected 6th page. In Program A, the entire file is brought into memory using <code>read</code> , the selected pages are modified, and the entire file is written out to disk using <code>write</code> . On the first 5 pages, Program B does an explicit disk seek to read the first integer. Then B does a seek and a write to modify the 6th page. | 15 |
| 1-2 | A third version of the C program from Figure 1-1, written using <code>mmap</code> | 16 |
| 1-3 | Concurrent processes sharing data through a memory mapped file. | 17 |
| 1-4 | An interleaving of instructions from the execution of the programs in Figure 1-3 that causes a data race. | 18 |
| 1-5 | Two different locking protocols for the program in Figure 1-2. Program D acquires a global lock, while Program E acquires a different lock for every page. For simplicity, only the body of the code is shown here. | 18 |
| 1-6 | Program F. This version of the program from Figure 1-2 is written with memory-mapped transactions. | 20 |
| 1-7 | An illustration of the possible space of programs that can concurrently access data on disk. | 21 |
| 2-1 | LIBXAC program that increments the first integer of a memory-mapped file. | 27 |
| 2-2 | A side-by-side comparison of the program in Figure 1-2 and the parallel version written with LIBXAC. | 28 |
| 2-3 | A recursive function that uses nested transactions. | 29 |
| 2-4 | A transaction that that accesses local variables inside a transaction. | 31 |
| 3-1 | Changes to the memory map for a simple transaction. | 39 |
| 3-2 | An example of a consistency tree. | 44 |
| 4-1 | A simple transaction that (a) reads from n consecutive pages, and (b) writes to n consecutive pages. | 51 |
| 4-2 | Average time per page to execute the transactions shown in Figure 4-1 on Machine 1. For each value of n , each transaction was repeated 1000 times. | 52 |
| 4-3 | The transactions in Figure 4-1 written with the advisory function. The transaction in (a) reads from n consecutive pages, the transaction in (b) writes to n consecutive pages. | 53 |
| 4-4 | Average time per page to execute the transactions shown in Figure 4-3 on Machine 1. For each value of n , each transaction was repeated 1000 times. | 54 |
| 4-5 | Concurrency Test A: Each transaction increments the first integer on a page 10,000 times. | 60 |
| 4-6 | Concurrency Tests B and C: Test B increments every integer on the page. Test C repeats the transaction in Test B 1,000 times. I omit the outermost for-loop, but as in Figure 4-5, each transaction is repeated 10,000 times. | 60 |
| 5-1 | An illustration of the Disk Access Machine (DAM) model. | 64 |

| | | |
|-----|---|----|
| 5-2 | The van Emde Boas layout (left) in general and (right) of a tree of height 5. | 65 |
| 5-3 | Machine 3: Time for k th most expensive insert operation. | 70 |
| B-1 | An example of a LIBXAC log file when transactions execute (a) on one process, and (b) on two processes. | 82 |
| C-1 | Machine 1: Distribution of Delay Times Between Successive <code>gettimeofday</code> Calls. . . | 86 |
| C-2 | Average time per page to execute the transactions shown in Figure 4-3 on Machine 2. For each value of n , each transaction was repeated 1000 times. | 87 |
| C-3 | Average time per page to execute the transactions shown in Figure 4-3 on Machine 3. For each value of n , each transaction was repeated 1000 times. | 88 |
| C-4 | Average time per page to execute the transactions shown in Figure 4-3 on Machine 4. For each value of n , each transaction was repeated 1000 times. | 89 |

List of Tables

| | | |
|------|---|----|
| 2.1 | The LIBXAC functions for nondurable transactions. | 26 |
| 4.1 | Processor speeds and time per clock cycle for the test machines. | 50 |
| 4.2 | Average # of clock cycles per page access for transactions touching 1024 pages, with and without the advisory function. Numbers are in thousands of cycles. Percent speedup is calculated as $100 \left(\frac{\text{Normal} - \text{With Adv}}{\text{Normal}} \right)$ | 53 |
| 4.3 | Number of clock cycles required to enter SIGSEGV handler, call <code>mmap</code> , and exit handler (average of 10,000 repetitions). | 55 |
| 4.4 | Clock cycles required to write to a page for the first time after memory mapping that page. Each experiment was repeated 5 times. | 56 |
| 4.5 | Clock cycles required for a <code>memcpy</code> between two 4K character arrays in memory. | 56 |
| 4.6 | Average # of clock cycles per page access for transactions touching 1024 pages. All numbers are in thousands of clock cycles. | 57 |
| 4.7 | Average Access Time (μs) per Page, for Transactions Touching 1024 Pages. | 58 |
| 4.8 | Time required to call <code>msync</code> and <code>fsync</code> on a 10,000 page file with one random page modified, 1000 repetitions. All times are in ms. | 59 |
| 4.9 | Concurrency tests for nondurable transactions. Times are μs per transaction. Speedup is calculated as time on 1 processor over time on 2 processors. | 59 |
| 4.10 | Concurrency tests for durable transactions. Times are milliseconds per transaction. | 61 |
| 5.1 | Performance measurements of 1000 random searches on static trees. Both trees use 128-byte keys. In both cases, we chose enough data so that each machine would have to swap. On the small machine, the CO B-tree had 2^{23} (8M) keys for a total of 1GB. On the large machine, the CO B-tree had 2^{29} (512M) keys for a total of 64GB. | 66 |
| 5.2 | Timings for memory-mapped dynamic trees. The keys are 128 bytes long. The range query is a scan of the entire data set after the insert. Berkeley DB was run with the default buffer pool size (256KB), and with a customized loader that uses 64MB of buffer pool. These experiments were performed on the small machine. | 67 |
| 5.3 | The time to insert a sorted sequence 450,000 keys. Inserting sorted sequence is the most expensive operation on the packed memory array used in the dynamic CO B-tree. | 67 |
| 5.4 | Changes in Code Length Converting B ⁺ -tree and CO B-tree to Use LIBXAC. | 68 |
| 5.5 | Time for 250,000 durable insertions into LIBXAC search trees. All times are in ms. Percent speedup is calculated as $\frac{100(t_1 - t_2)}{t_2}$, where t_1 and t_2 are the running times on 1 and 2 processors, respectively. | 69 |
| 5.6 | The % speedup of LIBXAC search trees over Berkeley DB. Percent speedup is calculated as $\frac{100(t_L - t_B)}{t_B}$, where t_L and t_B are the running times on the LIBXAC and the Berkeley DB tree, respectively. Speedup is t_1/t_2 | 69 |
| C.1 | Delay (in clock cycles) between successive calls to timer using <code>rdtsc</code> instruction, 10,000 repetitions. | 85 |
| C.2 | Delay between successive calls to <code>gettimeofday</code> (in μs), 10,000 repetitions. | 86 |
| C.3 | Timing data for entering SIGSEGV handler, calling <code>mmap</code> , and leaving handler, 10,000 repetitions. All times are processor cycles. | 90 |

| | | |
|------|---|----|
| C.4 | Clock cycles to do 1,000 calls to <code>memcpy</code> between two 4K character arrays in memory, 1,000 repetitions. times are in μs | 90 |
| C.5 | Average Access Time (μs) per Page, for Transactions Touching 1024 Pages. | 91 |
| C.6 | Timing data for calling <code>msync</code> and <code>fsync</code> on a 10,000 page file with a random page modified, 1000 repetitions. All times are in μs | 91 |
| C.7 | Time to write 10,000 pages to a file, 1,000 repetitions. All times are in μs | 91 |
| C.8 | Time to compute SHA1 and MD5 hash functions on a single page. All times are in thousands of clock cycles. | 92 |
| C.9 | Concurrency tests for nondurable transactions. Times are μs per transaction. | 93 |
| C.10 | Concurrency tests for durable transactions. Times are per transaction. | 94 |
| C.11 | Time to do 250,000 nondurable insertions into LIBXAC search trees. | 95 |
| C.12 | Time to do 250,000 durable insertions on a single process into the various search trees, with write-caches on the harddrives enabled. All times are in ms. | 95 |