“

I KNOW WHO I WAS WHEN I GOT UP THIS MORNING, BUT I THINK I MUST HAVE BEEN CHANGED SEVERAL TIMES SINCE THEN

”

Lewis Carroll, Alice’s Adventure in Wonderland and Through the Looking Glass

OLTP THROUGH THE LOOKING GLASS, AND WHAT WE FOUND THERE

Written by Stavros Harizopoulos, Daniel J. Abadi, Samuel Madden and Michael Stonebraker.

Presented by Hiral Patwa

PLAN

- Introduce the questions of the paper
- Measure the expenses of OLTP components
- Talk about SHORE and modifications made
- Implications for future OLTP systems
- Conclusion

INTRODUCTION

- Online Transaction Processing (OLTP) processes a large number of short on-line transactions.
- Used in data entry or information retrieval. Queries are simple, require sub-second response times and return relatively few words.
- Effectiveness measured by the number of transaction per second (tps) it processed.

INTRODUCTION

- Features of OLTP:
  - Disk-resident B-trees and heap files,
  - Locking based concurrency control,
  - Support for multi-threading,
  - Log-based recovery, and
  - Efficient buffer manager
- Created during1970’s – 1980’s.

INTRODUCTION

- Modern microprocessors are very fast, and the computation time for many OLTP-style transactions is measured in microseconds.

- Performance improvements and cost reduction in hardware suggests a number of alternate systems

MEASURING THE EXPENSES OF OLTP

AIM: To optimize the OLTP databases systems for main memory storage as well as other database variants so as to reach new benchmarks.

HOW: Take an open sources modern DBMS (SHORE) and benchmarked it on a subset of TPC-C benchmark.

After recording first results, modified the system by removing features that produced new benchmarks at each step.

Initial implementation (running on a modern desktop machine) ran about 640 tps. After all the modifications, which was a tiny kernel of query processing code, it could process about 12700 tps.

MEASURING THE EXPENSES OF OLTP

What was removed?

1) **Logging:** Keeping track of log records or the changes in database structures slows down the performance.

   It isn't required if recoverability is not needed, or if recoverability is provided by another node on the network.

2) **Locking:** OLTP follows two-phase locking system. Locks are held until the transaction is either completed or aborted. Not just until the lock is needed.

   This causes a considerable amount of overhead.

MEASURING THE EXPENSES OF OLTP

3) **Latching:** Multi-threading feature required to be latched to the data structures before they can be accessed.

   Single thread application have shown noticeable improvement performance.

4) **Buffer Management:** If the database is stored in the main memory, it need not access the pages through the buffer pool.

   Hence eliminating a huge amount of overhead on every record access.

MEASURING THE EXPENSES OF OLTP

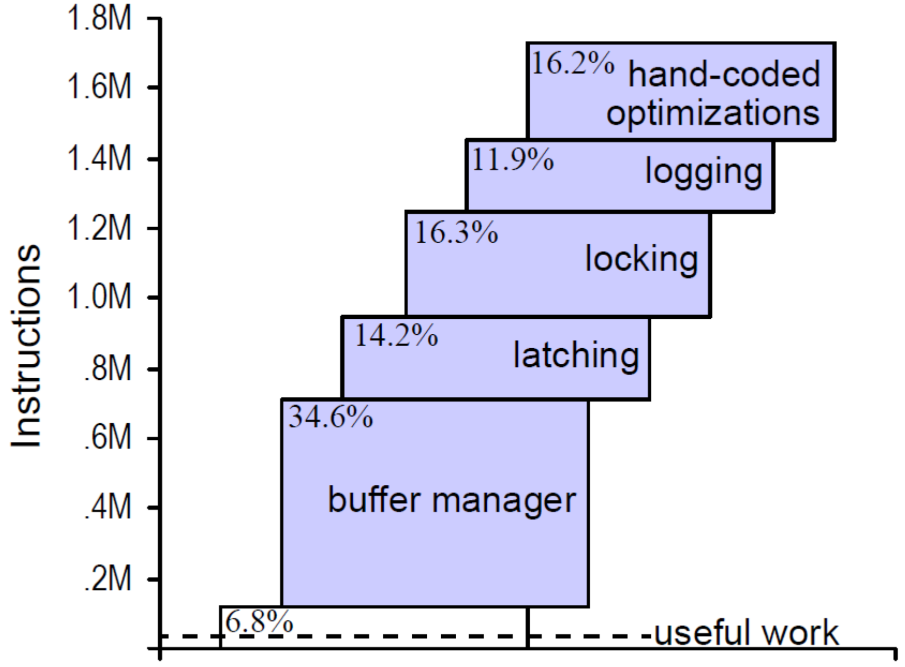
Results:

The left out kernel was a single-thread, lock-free, main memory DBMS with no recovery.

Each subsystem that was modified itself accounts for about 10-35% of the total runtime.

The actual instruction to process the query is "useful work", which is 1/60th of total runtime.

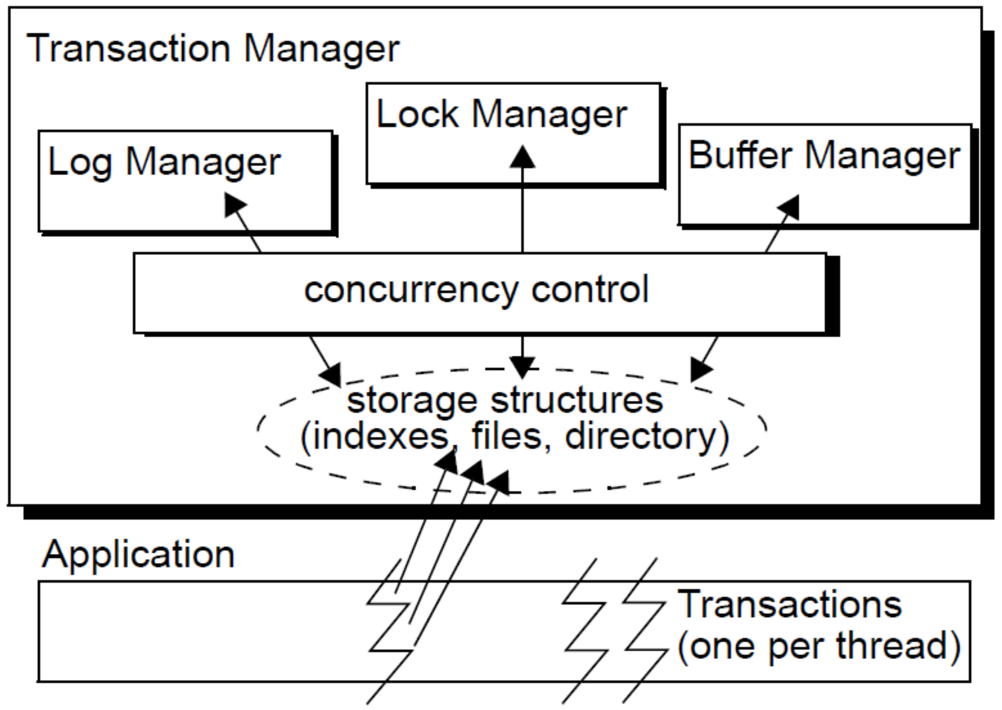
Measured through a minimal implementation built on top of the hand-coded optimizations



COMPONENTS IN SHORE

SHORE (Scalable Heterogeneous Object REpository) was designed at University of Wisconsin in the early 1990's.

It has a layered architecture



MODIFICATIONS TO SHORE

Due to the tight integrations of all managers in Shore, it was not possible to cleanly separate all the components, hence they were removed in an order.

The order was the following:

1. Remove logging
2. Remove locking OR latching
3. Remove latching OR locking
4. Remove code related to the Buffer Manager

MODIFICATIONS TO SHORE

Logging in SHORE

Features:

- Implements a write-ahead logging
- The logs are identified by sequence numbers Log Sequence Numbers(LSN)
- Needs a close interaction with log manager, buffer manager and transaction manager

Removed/Modification:

- Group commits were done and increase in the log buffer size
- Commented out the functions that were used to write and prepare the log records
- Addition of IF statements to avoid processing LSN’s

MODIFICATIONS TO SHORE

Locking in SHORE

This modification is interchangeable with removing Latching (as Latching is performed within Locking)

Features:

- As mentioned, it implements a two-phase locking system
- Each transaction tracks and lists all of the locks it holds
- Locks are logged when transaction enters the prepared state and released at the end of the transaction.

Removed/Modification:

- Changed the Lock Managers method to return immediately
- Call for request was returned as successful as if all the checks for locks were satisfied
- Changed the methods that were related to finding records, making them access them through B-tree indices.

MODIFICATIONS TO SHORE

Latching in SHORE

This modification is interchangeable with removing Locking (as Latching is performed within Locking)

Removed/Modification:

- Changed all mutex requests to be satisfied immediately
- Addition of IF-statements to avoid the requests for latching
- Replaced B-tree methods with methods that did not use latching

MODIFICATIONS TO SHORE

<u>Buffer Management in SHORE</u>

Features:

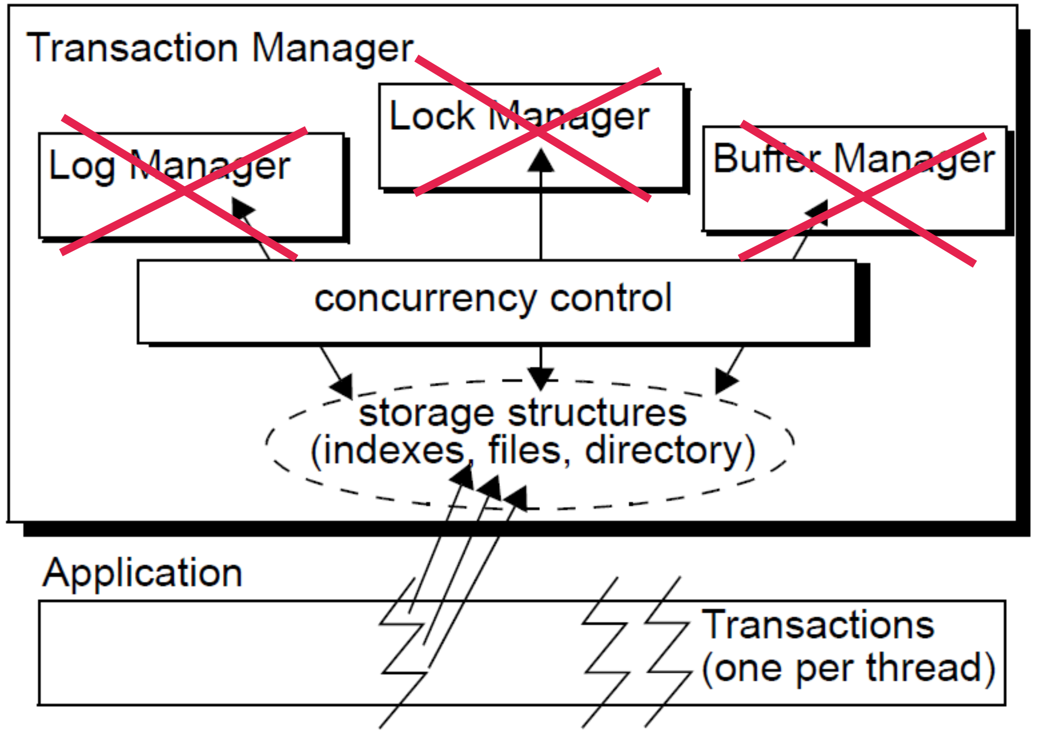
- Uses a fix method to read/write pages
- Latch to ensure consistency to fix method
- Reading a record is done by a method of pin/unpin (records are locked)
- Updates happen by copying the data to user's address space, changes are made there and then given to the storage manager

Removed/Modification:

- Abandoned the Shore's page allocation method and instead used the standard malloc library.
- Removal of page interface to buffer frames was tested, but never completed.

MODIFICATIONS TO SHORE

Shore after the modifications and removal:



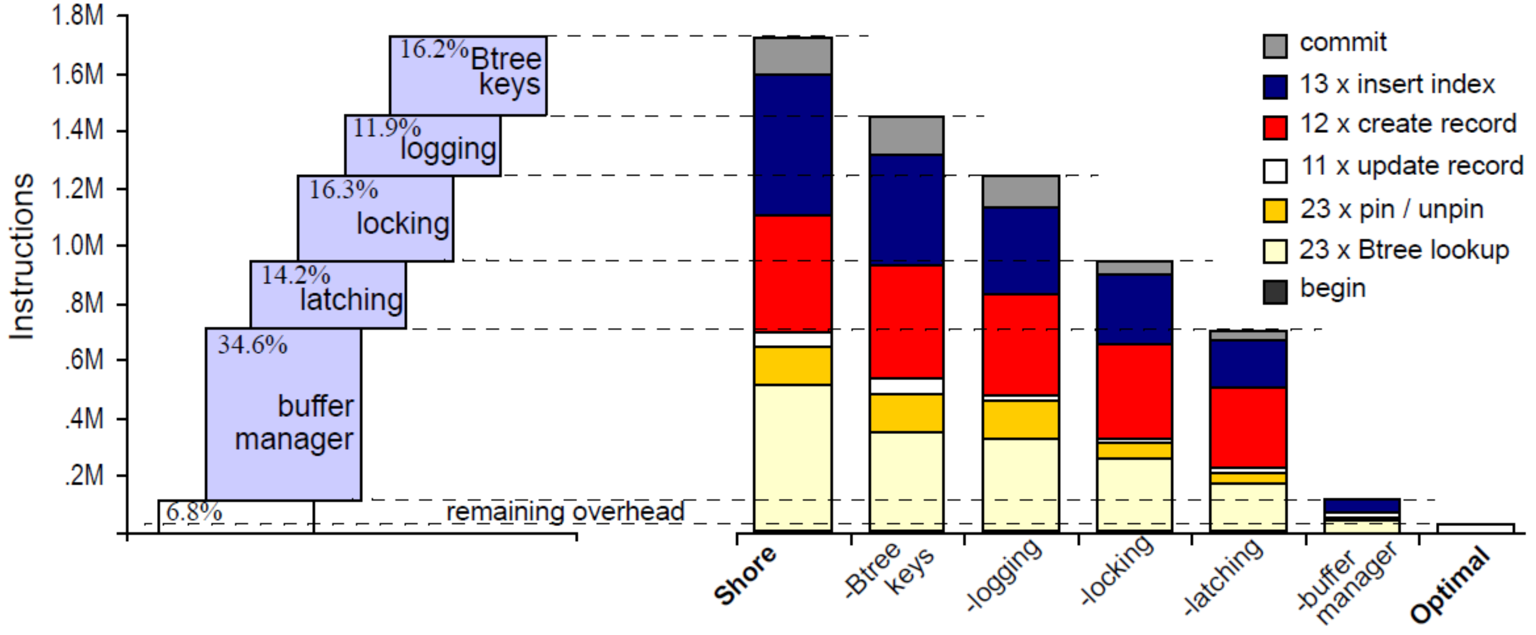
RESULT OF MODIFICATIONS TO SHORE

Testing of the modified Shore was done on a single-core Pentium 4, 3.2 GHz with 1MB L2 cache, hyper threading disabled, 1GB RAM; running on Linux 2.6

40K transaction of the New Order Transaction and Payments were run

Results were measured in:

1. Throughput (Time/#of transactions completed)
2. Instruction count



Detailed instruction count breakdown for New Order transaction

IMPLICATIONS FOR THE FUTURE

<u>Concurrency Control</u>

Transactions-at-a-time processing allow concurrency control to be turned off.

There are many DBMS applications that do not work well with single threaded transactions.

Optimistic concurrency control as an option?

IMPLICATIONS FOR THE FUTURE

Multi-core Support

Since multiple transaction concurrently on multiple cores require latching we think of other options.

1. Use virtualizations, and make it seem like each core is a single-threaded machine.

2. Attempt to exploit intra-query parallelism. However, the amount of intra-query parallelism available in an OLTP transaction is likely to be limited.

IMPLICATIONS FOR THE FUTURE

Replication Management

A typical OLTP system uses “active-passive” replication which requires two-phase commits. This results in high overheads.

To counteract, an active-active replication is considered. All replicas are active and the transaction is processed synchronously on all replicas

IMPLICATIONS FOR THE FUTURE

Weak Consistency

Eventual Consistency is mostly favored in traditional OLTP DBMS over strong consistency,

Eventual consistency to be valid, transactional consistency needs to be under a general workload.

This would be an interesting exercise, since the studies so far suggest that removing transactional support, locking and logging, from the main memory system could yield a very high performance system.

IMPLICATIONS FOR THE FUTURE

Cache-conscious B-trees

In this paper, no conversion of B-trees to a cache-conscious format was done as it would not have a major impact after all the optimizations has been made.

The idea is to target the cache misses in the B-tree code that may be a new bottleneck AFTER all stripping the system down to the minimal-overhead kernel.

There may be a case, where other indexing structures such as Hash Tables, might perform better.

CONCLUSION

- Most significant overhead contributors were found to be:
    1. Buffer Management
    2. Locking
    3. Logging
    4. Latching

- Unless we optimize the system, the performance of a main-memory database is unlikely to be much better than the conventional database.

- This paper also confirms that recent proposals for stripped down systems may have something solid to work on for.

“

CURIOUSER AND CURIOUSER.

”

Lewis Carroll, Alice’s Adventure in Wonderland and Through the Looking Glass