OLTP – Through the Looking Glass, and What We Found There

Authors :Stravos Harizopoulous, Daniel Abadi, Samuel Madden, Micheal Stonebraker

Presentation : Amit Tiwari

Using the Potential

- Many OLTP databases now fit in main memory
- Most database transactions are processed in milliseconds or less
- Network clusters have aggregated memory measured in hundreds of gigabytes

Database architecture has changed very little

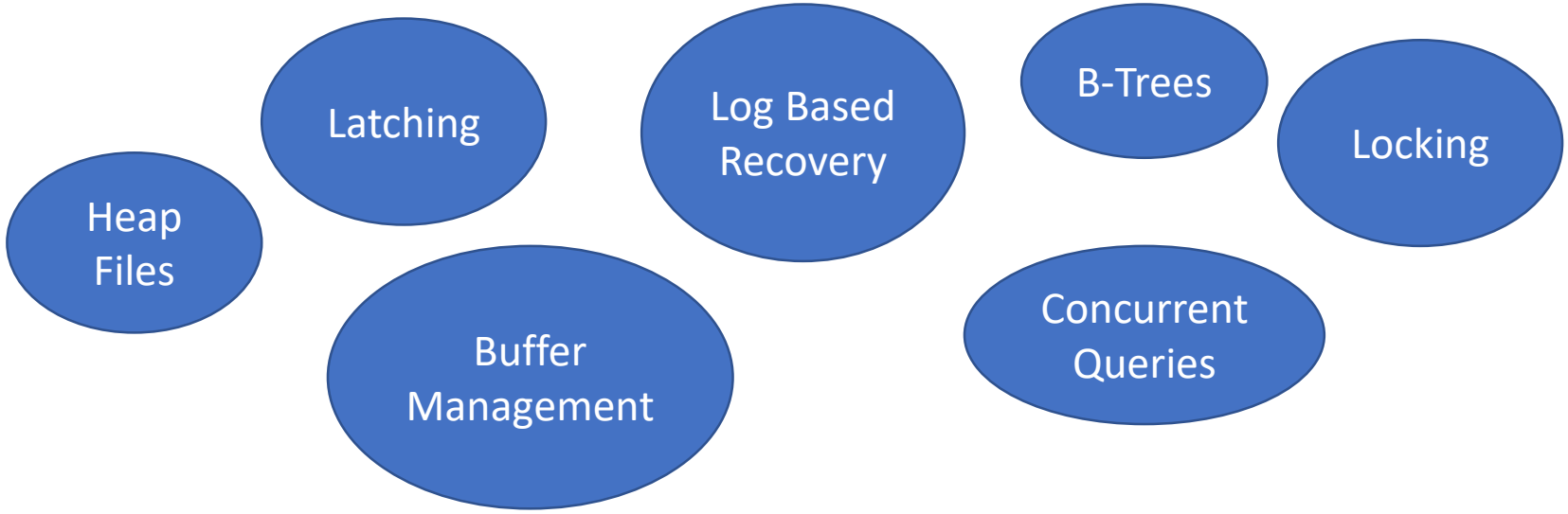
Perspectives From a Looking Glass

"Now, here, you see, it takes all the running you can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!"

Lewis Carroll, Alice Through the Looking Glass

OLTP

- A collection of on-disk data structures for table storage



- We need everything at all the time?

DBMS - Alternate versions

- Logless Database
  - E.g. Harp, Harbour, C-Store
- Single Threaded Database
  - Is virtualization an answer to this?
- Transaction-less Databases
  - Light weight forms of transactions where all reads are done before writes

Plan of Presentation

IDENTIFY	MODIFY	MEASURE	INFER
Where is the time most consumed?	Modify an existing database	Measure performance gains	What can we learn from this?

*database like systems are databases without the full suite of database features and have varying forms of consistency, reliability, concurrency etc.

What could be the Bottlenecks? OLTP Trends

- Cluster Computing – shared nothing architecture
  - Need to build the system ground up
  - E.g. Gamma

- Memory Resident Databases
  - Many OLTP systems already fit especially in the aggregate main memory of a large cluster
  - Disk-Optimized tuple formats and page layouts (or lack)

What could be the Bottlenecks? OLTP Trends

- Single Threading in OLTP Systems
  - If databases are memory resident there is never any disk waits
  - For production transaction systems, transactions are executed through stored procedures
  - How about multiple processors ? Or network latency ?

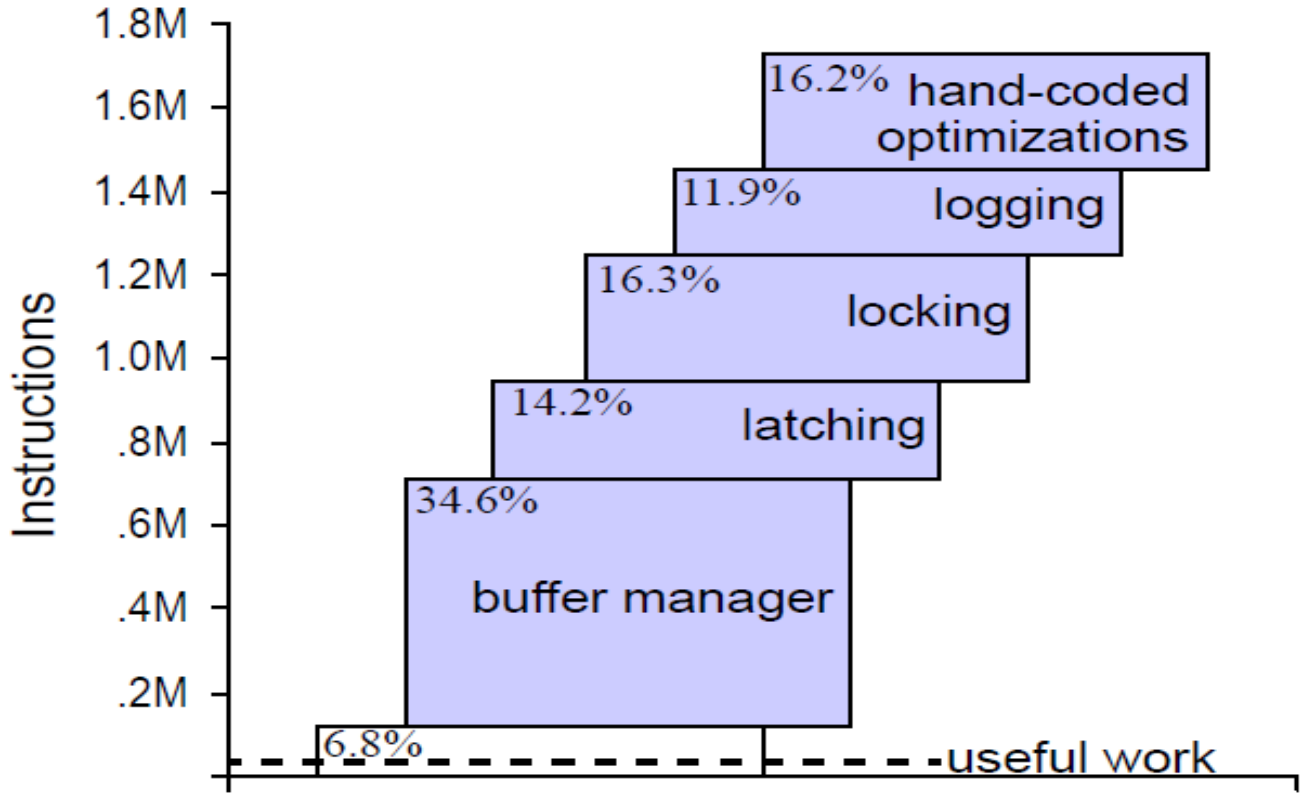
- High Availability and Logging
  - We tend to use two or more times the hardware for this
  - Copy missing states from other database replicas?

What could be the Bottlenecks? OLTP Trends

Other Variants

- Eventual Consistency Guarantees
  - The belief is that availability is more important than transactional semantics
- Less logging with things like snapshot isolation
  - Users ready to trade transactional semantics for performance
- Some forms of transactions require less machinery
  - E.g. if all the transactions are two phase and are guaranteed not to abort after completion of reads

Where are the Bottlenecks?



Instruction count of various components for the new order transaction from TPC-C

Where are the bottlenecks?

- Logging
  - Assembling log records and tracking down all changes slows performance
  - Recoverability not always a requirement or can be provided alternately
- Locking
  - Considerable overhead as all accesses to database are governed by Lock Manager
- Latching
  - Like locking it has considerable overhead.
  - Single threaded application have shown noticeable performance improvement
- Buffer Management
  - Memory pages accessed through a buffer pool contribute to overhead

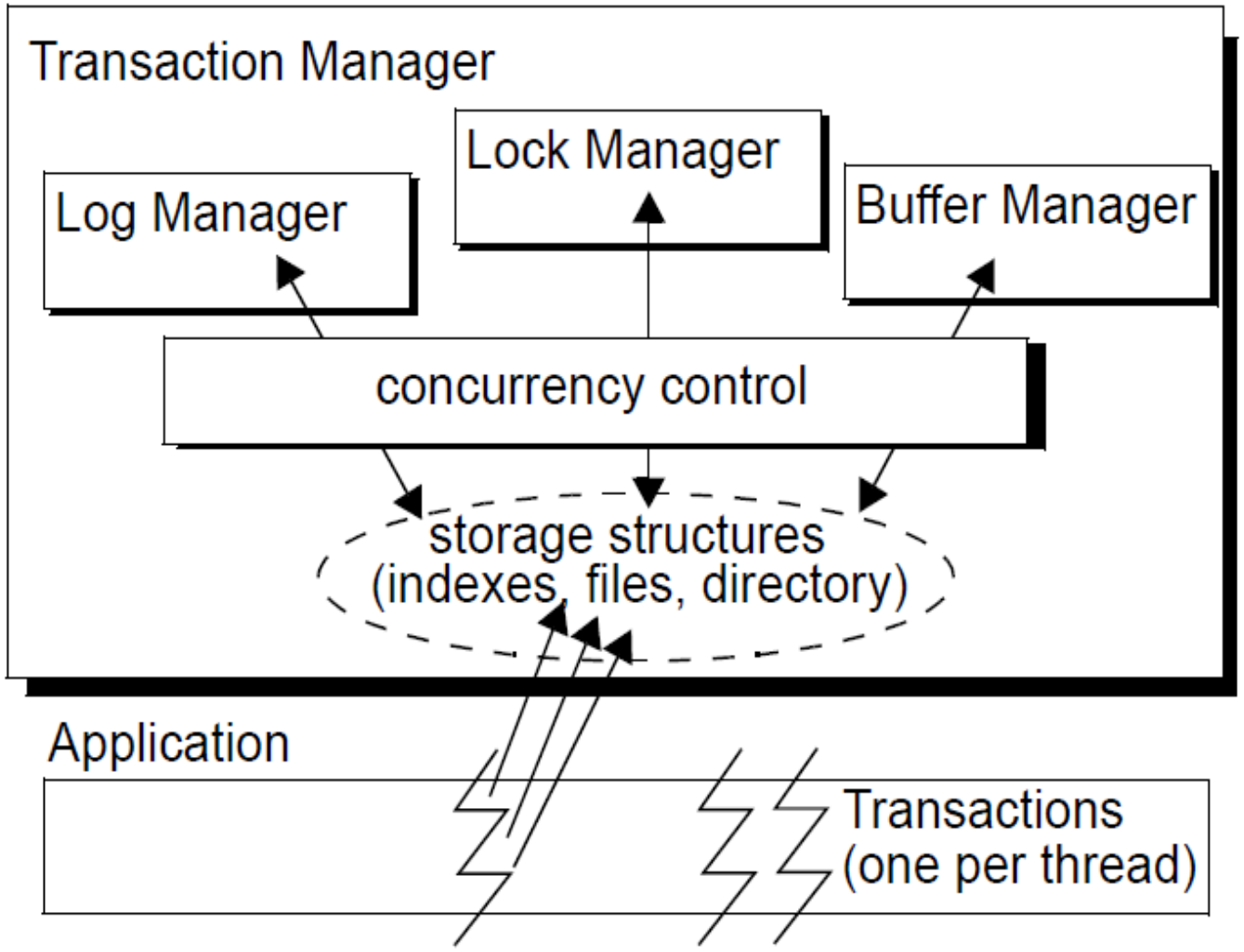
What We Learnt So Far

IDENTIFY	MODIFY	MEASURE	INFER
	Modify an existing database	Measure performance gains	What can we learn from this?

*for modify phase we use an open source database called SHORE

SHORE – Components

- Scalable Heterogenous Object Repository
- Has a layered architecture that allows users to choose the appropriate level of support for their applications from several components
- Has all the features found in all modern DBMS
- Is provided as a library and the user code is linked against the library



What We Look To Remove

OLTP Properties & New Platforms	DBMS Modification
Logless Architecture	Remove Logging
Partitioning, Commutativity	Remove locking (when applicable)
One transaction at a time	Single thread, remove locking, remove latching
Main memory resident	Remove buffer manager, directory
Transaction-less database	Avoid transaction bookkeeping

Components in SHORE and Their Removal

- Logging Features
  - Implements a write ahead logging
  - Logs are identified by sequence numbers - LSN
  - Requires close interaction with log manager, buffer manager and transaction manager

- Logging Removal
  - Doing group commits and increasing the log buffer size
  - Commenting out functions used to prepare and write log records
  - Adding IF statements at code level to avoid processing LSNs

Components in SHORE and Their Removal

- Locking Features
  - Implements standard two-phase locking
  - Each transaction keeps a list of locks it holds
  - Locks are logged when transactions enter prepared state and released at end

- Locking Removal
  - Lock Manager Methods were changed to return immediately
  - The call for request was returned successful as if all checks for locks were satisfied
  - Methods related to finding records , finding them through B-tree indexes modified

- Latching Removal
  - Changed mutex requests to be immediately satisfied
  - Adding IF statements to avoid requests for latching
  - Replacing B-Tree methods with one that didn’t use latching

Components in SHORE and Their Removal

- Buffer Manager Features
  - Used to read/write pages, uses the fix method
  - There is a latch to ensure consistency to fix method
  - Reading a record is done by a pin/unpin method (records are locked)
  - For updates data is copied to user's address space, changes made there and then given to storage manager

- Buffer Manager Removal
  - Removing page allocation mechanism and using malloc instead
  - Removal of page interface to buffer frames was tried but not completed
  - Since the page allocation was removed, the buffer manager became 'thin'

What We Learnt So Far

IDENTIFY

MODIFY

MEASURE

Measure performance gains

INFER

What can we learn from this?

*measurements performed on a single core Pentium 4 3.2GHz, 1MB L2 Cache, Hyperthreading disabled, 1GB RAM, Linux 2.6

Performance Study – TPC-C

- Designed to represent any industry that manage, sell or distribute a product or service
- Designed to scale with suppliers and warehouses
- Database size for one warehouse is approximately 100 MB , a total five warehouses are used
- Involves five concurrent transactions of different types and complexity
  - Entering orders
  - Recording Payments
  - Delivering Orders
  - Checking the status of orders
  - Monitoring the level of stock at the warehouse

Performance Study – Workload

- A new order transaction places an order for 5-15 items with 90 percent supplied from local warehouses and 10 percent from remote warehouses
- Variations are avoided by fixing 10 items taken from local warehouses
- Payment updates the customer's balance and generates a history record

New Order

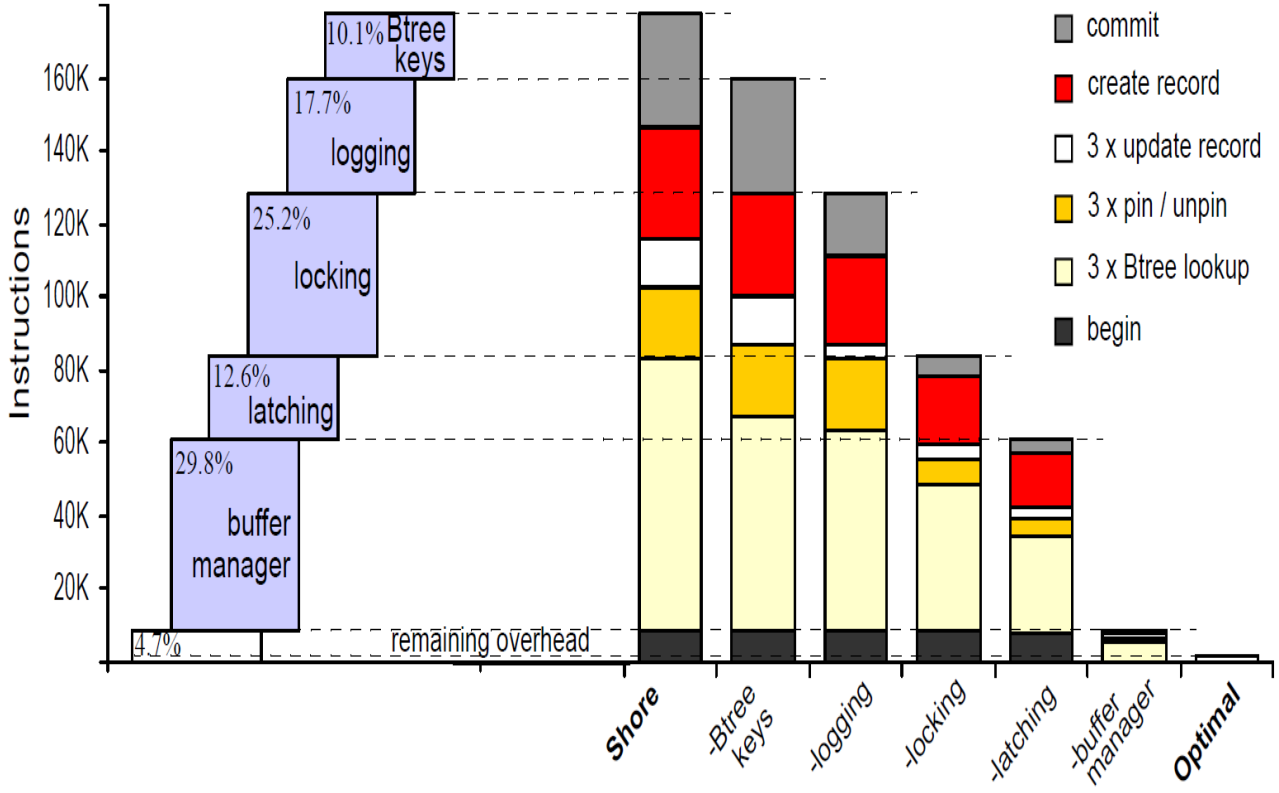
```
begin
for loop(10)
.....Btree lookup(I), pin
Btree lookup(D), pin
Btree lookup (W), pin
Btree lookup (C), pin
update rec (D)
for loop (10)
.....Btree lookup(S), pin
.....update rec (S)
.....create rec (O-L)
.....insert Btree (O-L)
create rec (O)
insert Btree (O)
create rec (N-O)
insert Btree (N-O)
insert Btree 2ndary(N-O)
commit
```

Payment

```
begin
Btree lookup(D), pin
Btree lookup (W), pin
Btree lookup (C), pin
update rec (C)
update rec (D)
update rec (W)
create rec (H)
commit
```

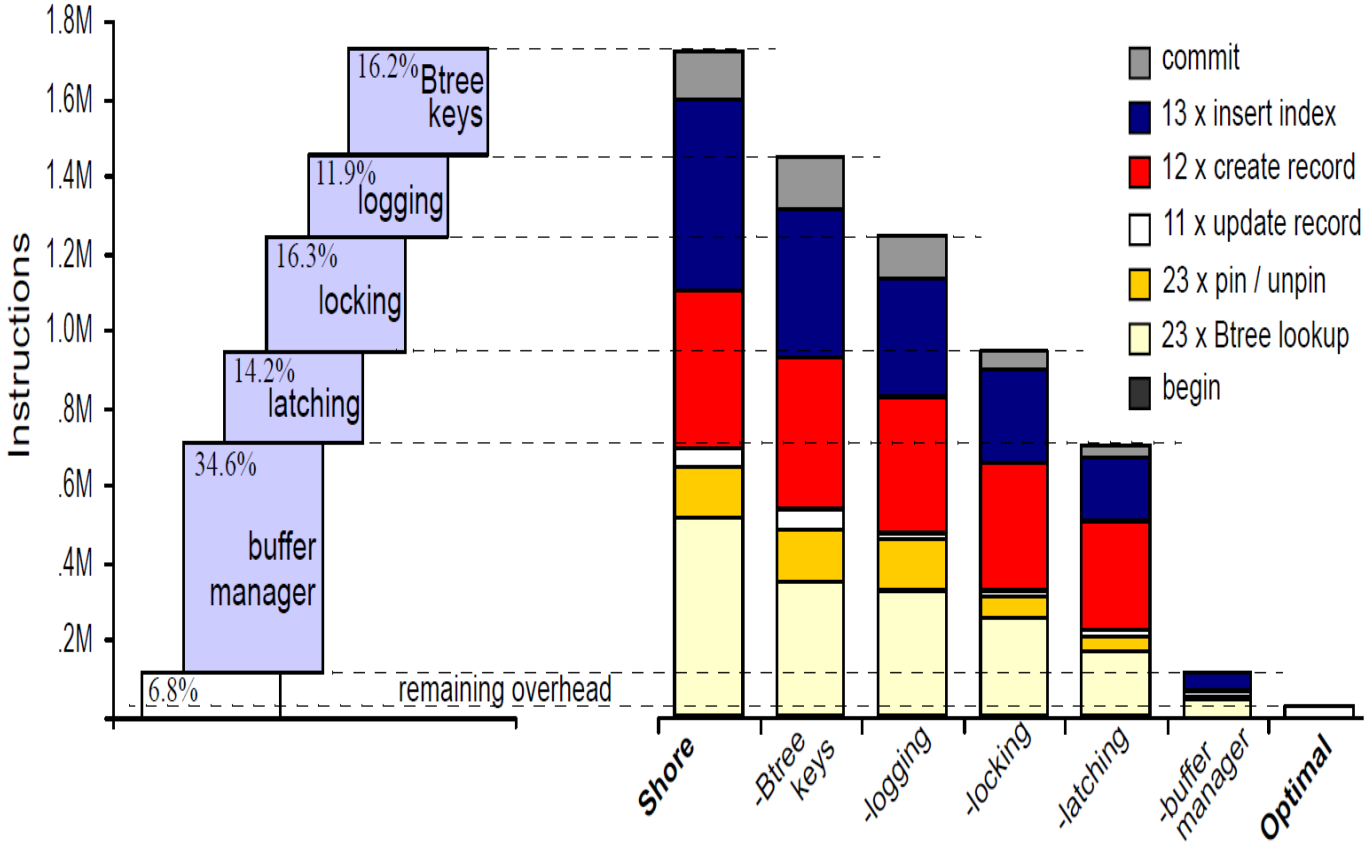
Performance Study – Details (Payment)

- B-Tree – 18 % of total instruction count
- Locking - 25 % of total instruction count, has a lot of effect on pin/unpin operations
- Latching – 13% of total instruction count, effects create record and B-Tree lookups
- Buffer Manager – 30% of total instruction count, changes to page lookups make gains
- The remaining kernel requires only 5% of total instructions for a 20X performance gains



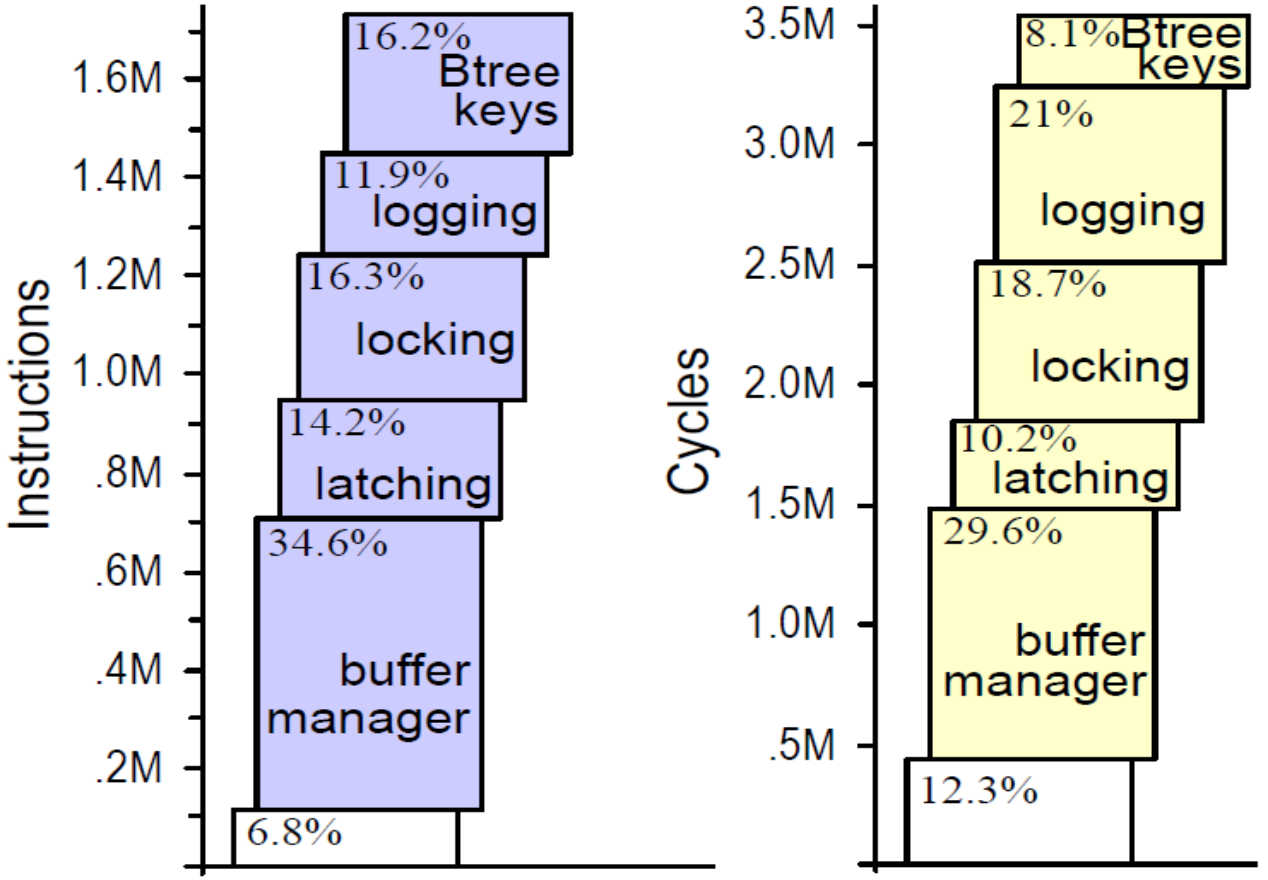
Performance Study – Details (New Order)

- About 10 times as many instructions as payment
- B-Tree – 16 % of total instruction count
- Logging - 12 % of total instruction count, has a lot of effect on pin/unpin operations
- Locking – 16% of total instruction count, effects create record and B-Tree lookups
- Buffer Manager –14% reduction of total instruction count, reduction in page allocations and B-Tree depths



Performance Study – Instructions vs Cycles

- The two may not be identical
- Cache misses and pipeline stalls (due to branching) can cause some to take more instructions than others
- The residual kernel uses a larger fraction of cycles , because it is branch heavy, has system calls
- Logging uses more cycles as it touches main memory , creating and writing log records



Performance Study - Throughput

- After all the optimizations, CPU cycles dominate
- Average : 80 microsecond/transaction or 12,700 transactions/second
  - Initially it was 640 transactions/second
- Useful Work – only about 22 microseconds/transaction
  - Kernel is not entirely devoid of system calls of SHORE
- Significant improvements with log flushing: 1700/second
- About 20 percent improvement in throughput

What We Learnt So Far

IDENTIFY

MODIFY

MEASURE

INFER

What can we learn from this?

Inferences

- Concurrency Control
  - Large gains when concurrency control is turned off
  - Is optimistic concurrency control one of the answers?

- Multi Core Support
  - Multiple transaction concurrently on separate cores within a single site, requires latching
  - Need to revisit new implementations of latching, can we reduce the overhead?
  - Virtualization can be an alternative, no performance metrics available
  - Intra-query parallelism may be a feasible option

Inferences

- Replication Management
  - Active-Passive replication requires two phase commits, generates overhead
  - Move towards active-active replication, can be done for memory resident systems
- Consistency
  - Eventual consistency can be favoured over strong consistency
  - But for eventual consistency we need transactional consistency under a general workload
- Cache Conscious B-Trees
  - Not as big of a bottleneck as other components
  - If however the system is stripped to a basic kernel, it may be significant

What We Learnt So Far

IDENTIFY

MODIFY

MEASURE

INFER

Conclusions

"I believe that "one size does not fit all". i.e. in every vertical market I can think of, there is a way to beat legacy relational DBMSs by 1-2 orders of magnitude." **— Mike Stonebraker.**

- Significant gains occur when multiple optimizations are applied
- The benefits of stripping out any one component has only a small benefit

Influences on Database Systems