Main Memory Storage Engines

Justin A. DeBrabrant
debrabrant@cs.brown.edu
Roadmap

• Paper 1: Data-Oriented Transaction Execution
• Paper 2: OLTP Through the Looking Glass
• Paper 3: Generic Database Cost Models for Hierarchical Memory Systems
Storage Engine?

• the part of the database that actually stores and retrieves data
  – responsible for db performance
    • concurrency, consistency
  – separate from the database “front end”

• A single database can have several database engine options
  – e.g. MySQL supports InnoDB and MyISAM
Paper 1

• Data Oriented Transaction Execution
  – I. Pandis at al. (CMU/EPFL/Northwestern)
  – VLDB ‘10
Motivation

• Hardware has changed
  – recently, we’ve run into “thermal wall”
    • hard to fit more resistors per chip
    • …must abide by Moore’s Law!
      – add more cores per chip
      – rely on thread-level parallelism
  – most current architectures designed in the 80’s
    • what assumptions were made about the hardware?
Thread-to-Transaction Model

- in most database engines, each transaction assigned to its own thread
  - more cores = more parallel threads
  - each thread responsible for locking shared resources as needed
    - works fine with a few threads, how about thousands executing concurrently on hundreds of hardware contexts?
Data Access Pattern

• Each thread only worries about its own transaction
  – no coordination among transactions
    • i.e. uncoordinated data access
  – leads to high lock contention, especially at data “hot spots”
Data Access Visualization

Thread-to-transaction (Conventional)

![Graph showing thread-to-transaction data access visualization](image)

- **Thread-to-transaction (Conventional)**: A scatter plot represents the access pattern over time. Each dot corresponds to an access event, with the x-axis representing time in seconds and the y-axis representing the number of districts accessed. The visualization helps in understanding the data access patterns and can be used to optimize data access strategies.

**Legend**:
- **DISTRICTS**: Axes indicating the number of districts accessed.
- **Time (secs)**: Time in seconds along the x-axis.

**Graph Analysis**
- **Data Trends**: The graph displays a dense distribution of access events, indicating a high frequency of data accesses.
- **Access Patterns**: The clustering of dots suggests regions with high access density, which could be critical for performance optimization.

**Conclusion**
- **Data Insights**: Understanding these patterns is crucial for optimizing database systems and hardware configurations.
- **Optimization Strategies**: By analyzing these access patterns, systems can be designed to reduce latency and improve efficiency.
This paper makes three contributions.

1. Contributions and document organization

Also, it substitutes the centralized approach and eliminates the contention on the lock manager (Figure 2(a)).

Utilized, running Nokia’s TM processing system and a prototype DORA implementation, a single-threaded SHORE storage manager [3] running two systems. Harizopoulos et al. [9] analyze the behavior of the state-of-the-art storage manager.

A conventional system could not handle the problem of contention. Neither Rdb/VMS nor Cache Fusion handle the problem of contention. However, neither of them distributes the logical locks. Instead, they do not physically partition the database system design. In contrast, DORA ensures that the database system design exhibits predictable access flow from one thread to the other.

A system adopting thread-to-data assignment can exploit the regular pattern of data accesses to its subset of database records. The exclusive lock is acquired to allow shared-disk clusters to combine their buffer pools and execute the transaction. Without admission control the performance benefits for smaller actions according to Section 5. Each thread coordinates the read holds an exclusive lock. Figure 2(b) shows the contention within the lock manager.

DORA compared to Baseline when the workload consists of Order-Status transactions [20]. The DORA prototype utilizes running Nokia’s TM processing system and a prototype DORA implementation.

A benchmark [19] and TPC-C exhibit predictable access flow from one thread to the other. Section 3 explains why a lightweight thread-local lock is utilized. Running Nokia’s TM processing system and a prototype DORA implementation, a single-threaded SHORE storage manager [3] running two systems. Harizopoulos et al. [9] analyze the behavior of the state-of-the-art storage manager.

A conventional system could not handle the problem of contention. Neither Rdb/VMS nor Cache Fusion handle the problem of contention. However, neither of them distributes the logical locks. Instead, they do not physically partition the database system design. In contrast, DORA ensures that the database system design exhibits predictable access flow from one thread to the other.

A system adopting thread-to-data assignment can exploit the regular pattern of data accesses to its subset of database records. The exclusive lock is acquired to allow shared-disk clusters to combine their buffer pools and execute the transaction. Without admission control the performance benefits for smaller actions according to Section 5. Each thread coordinates the read holds an exclusive lock. Figure 2(b) shows the contention within the lock manager.

DORA compared to Baseline when the workload consists of Order-Status transactions [20]. The DORA prototype utilizes running Nokia’s TM processing system and a prototype DORA implementation.

A benchmark [19] and TPC-C exhibit predictable access flow from one thread to the other. Section 3 explains why a lightweight thread-local lock is utilized. Running Nokia’s TM processing system and a prototype DORA implementation, a single-threaded SHORE storage manager [3] running two systems. Harizopoulos et al. [9] analyze the behavior of the state-of-the-art storage manager.

A conventional system could not handle the problem of contention. Neither Rdb/VMS nor Cache Fusion handle the problem of contention. However, neither of them distributes the logical locks. Instead, they do not physically partition the database system design. In contrast, DORA ensures that the database system design exhibits predictable access flow from one thread to the other. A system adopting thread-to-data assignment can exploit the regular pattern of data accesses to its subset of database records. The exclusive lock is acquired to allow shared-disk clusters to combine their buffer pools and execute the transaction. Without admission control the performance benefits for smaller actions according to Section 5. Each thread coordinates the read holds an exclusive lock. Figure 2(b) shows the contention within the lock manager.

DORA compared to Baseline when the workload consists of Order-Status transactions [20]. The DORA prototype utilizes running Nokia’s TM processing system and a prototype DORA implementation.
The future looks bleak…

• Not quite!

• Idea: “Coordinate” data access patterns
  – rather than having threads contending for locks, have transactions contending for threads
  – distribute the transactions to the data, not data to the transactions
Thread-to-Data Model

- each thread is coupled with a disjoint subset of the database
- threads coordinate access to their own data using a private locking mechanism
"Coordinated" Data Access

![Thread-to-data (DORA)](image)
A Data Oriented Architecture (DORA)

- a shared-everything architecture designed to scale to very high core counts
- retains ACID properties
- data (i.e. relations) are divided into disjoint datasets
  - 1 executor (thread) per dataset
Routing

• How to map datasets?
  – use a routing rule

• Routing rules use a subset of columns from a table, called *routing fields*, to map rows to datasets
  – in practice, columns from primary or candidate keys are used
  – can be dynamically updated to balance load
Transaction Flow Graphs

• used to map incoming transaction to executers
• *actions* are the data access parts of the query
• *identifiers* describe which columns an action uses
• What about actions that don’t match routing fields?
  – called *secondary actions*, more difficult
Secondary Actions

• which executer is responsible?
  – for indexes that don’t index the routing fields, store the routing fields in the leaf nodes
    • added space overhead?
    • expensive to update indexes if routing fields are changed?
Rendezvous Points

• often, data dependencies exist between actions
  – insert *rendezvous points* between actions with data dependencies
    • logically separates execution into different phases
    • system cannot concurrently execute actions from different phases
Executing an Action

• 3 structures:
  – incoming action queue
    • processed in order received
  – completed action queue
  – thread-local lock table
    • use action identifiers to “lock” data to avoid conflicts
Inserts and Deletes

- Still need to acquire row-level locks through centralized locking manager
  - why?
    - T1 deletes a record
    - T2 inserts a record into the slot vacated by the record deleted by T1
    - T1 aborts but can’t roll back, slot is taken
  - row-level locks often not a source of contention
Experimental Setup

• 3 benchmarks used, all OLTP
  – TM-1
    • 7 transactions, 4 with updates
  – TPC-C
    • 150 warehouses (approx. 20 GB)
  – TPC-B
    • 100 branches (approx. 2 GB)
Lock Contention

The bar charts show the distribution of different types of locks (Thread-Local, Row-level, and Higher-level) across various benchmarks (TM1 Base, TM1 DORA, TPC-B Base, TPC-B DORA, OrdSt Base, OrdSt DORA) and workloads (OrderStatus, TPC-B, TPC-C). The y-axis represents the number of locks acquired per 100 transactions.

The results indicate that DORA has minimal interaction with the centralized lock manager, as seen in Figure 5. DORA achieves a small overhead, approximately 5%, which is much smaller than the overhead incurred by the conventional system. Figure 6 confirms that DORA reduces the number of contended locks and improves performance, especially in heavy contention scenarios.

In typical OLTP workloads, such as those in the TM1 benchmark, the conventional system suffers from contention in the lock manager. However, DORA exploits intra-transaction parallelism and reduces contention, resulting in improved performance.

The diagrams also show that DORA outperforms the conventional system in terms of scalability. As the load in the system increases, the performance of the conventional system decreases, while DORA maintains a high performance level. This is particularly evident in the case of TM1, where DORA demonstrates a significant advantage over the conventional system, even under heavy load conditions.

In summary, DORA provides a significant improvement in scalability and performance compared to the conventional system, especially in scenarios with heavy contention and high load.
First, we examine the impact of contention on the lock relative behavior between the needed profiling, we used tools from Sun Studio 12 suite. The using the Sun CC v5.10 compiler. For measurements that less than 5%. We use the highest level of optimization options multiple times, and the measured relative standard deviation is increasing number of hardware resources. The workload for those are uncontended. It is system suffers from contention in the lock manager. That is of the DORA mechanism is small. Much smaller than the execution. In contrast, for DORA for the Baseline system, growing to see that the contention in lock manager becomes the bottleneck ck manager becomes the bottleneck systems as the CPU utilization increases. The other two graphs show the time breakdown for each of the two systems. We can systems as the CPU utilization increases. The other two graphs show the time breakdown for each of the two systems. We can
Response Times

![Bar Chart]

- **GetNewDest**: Baseline is slightly higher than DORA.
- **UpdSubData**: DORA is significantly lower than Baseline.
- **NewOrder**: DORA is significantly lower than Baseline.
- **Payment**: DORA is significantly lower than Baseline.
- **TPC-B**: DORA is significantly lower than Baseline.

**Legend**
- Baseline
- DORA

**Axes**
- **Norm. Response Time**
- **Transactions**:
  - GetNewDest
  - UpdSubData
  - NewOrder
  - Payment
  - TPC-B
Conclusions

- Traditional database engines not made for the amount of thread-level parallelism seen in machines today
  - lock contention a major part of that
- A thread-to-data approach can significantly reduce lock contention
Paper 2

- OLTP Through the Looking Glass, and What we Found There
  - Stavros Harizopoulos et al.
  - SIGMOD ‘08
Motivation

• Hardware has changed
  – db systems were designed when memory was sparse
  – many OLTP databases can fit entirely in memory

• Even in memory, there are other bottlenecks
  – logging, latching, locking, buffer management
Alternative Architectures

• logless
  – removing logging

• single transaction
  – remove locking/latching

• main memory resident
  – remove transaction bookkeeping
Goals

• Remove each of the “unnecessary” parts, one by one, and evaluate performance
  – Determine relative performance gains by removing each feature
Instruction Count Breakdown

Figure 5. Detailed instruction count breakdown for Payment transaction.
Figure 6. Detailed instruction count breakdown for New Order transaction.
Conclusions

• Antiquated disk-based features can cause significant overhead in a main memory system

• Each component of a system should be carefully evaluated
Paper 3

• Generic Database Cost Models for Hierarchical Memory Systems
  – S. Manegold et al.
  – VLDB ‘02
Motivation

• Cost models are a key part of query optimization
  – traditional cost models based on disk accesses
• What about in a main memory system?
  – memory hierarchy
    • L1, L2, L3, main memory, (solid-state?)
Goals

• An accurate cost model should weight each memory hierarchy differently
  – overall “cost” of an operator should be the sum of the cost at all memory hierarchies
  – each level has different access cost
    • weight each access by that level’s cost
Data Access Patterns

• different operators exhibit different data access patterns
  – pattern dictates both cost and number of cache misses

• How to accurately model access patterns?
  – basic access patterns
    • single/repetitive sequential traversal, single/repetitive random traversal, random
    • compound access patterns
Cost Models

• For each basic access pattern, derive custom cost model (not shown)
• Combine basic access pattern cost models to derive compound access pattern cost models
• For each database operator (i.e. sort), map to a cost model
Experimental Analysis

**Figure 1:**

- **a)** Quick-Sort
- **b)** Merge-Join
- **c)** Hash-Join

**Graphs:**
- **IIUII=C2**
- **IIHII=C3**

**Legend:**
- L1 misses
- L2 misses
- TLB misses
- time [ms]
Conclusions

• Basic cost models presented can model the costs in main memory systems
• These memory-based cost models could also be used to enhance current disk-based cost models
Questions?