Main Memory Storage Engines

Justin A. DeBrabant
debrabant@cs.brown.edu

BROWN

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

BROWN

Paper 1

- Data Oriented Transaction Execution
  - I. Pandis at al. (CMU/EPFL/Northwestern)
  - VLDB '10

BROWN

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?

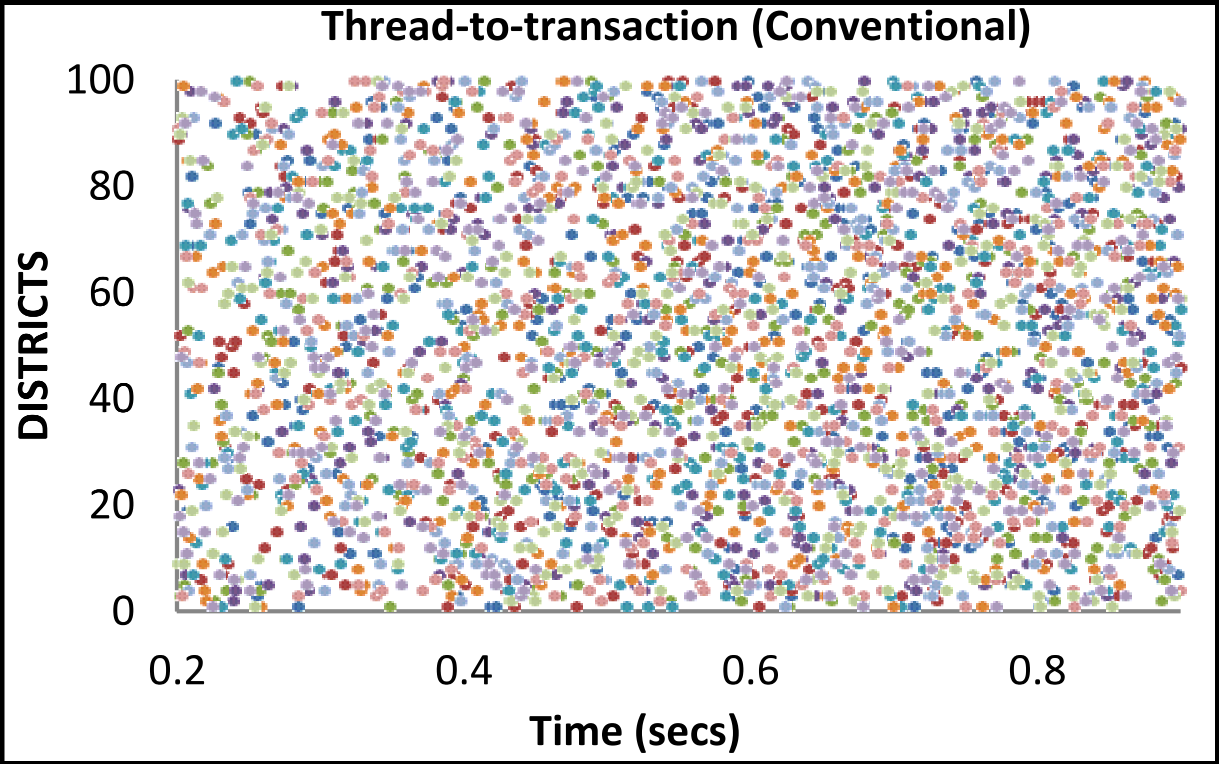
Thread-to-Transaction Model

- 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

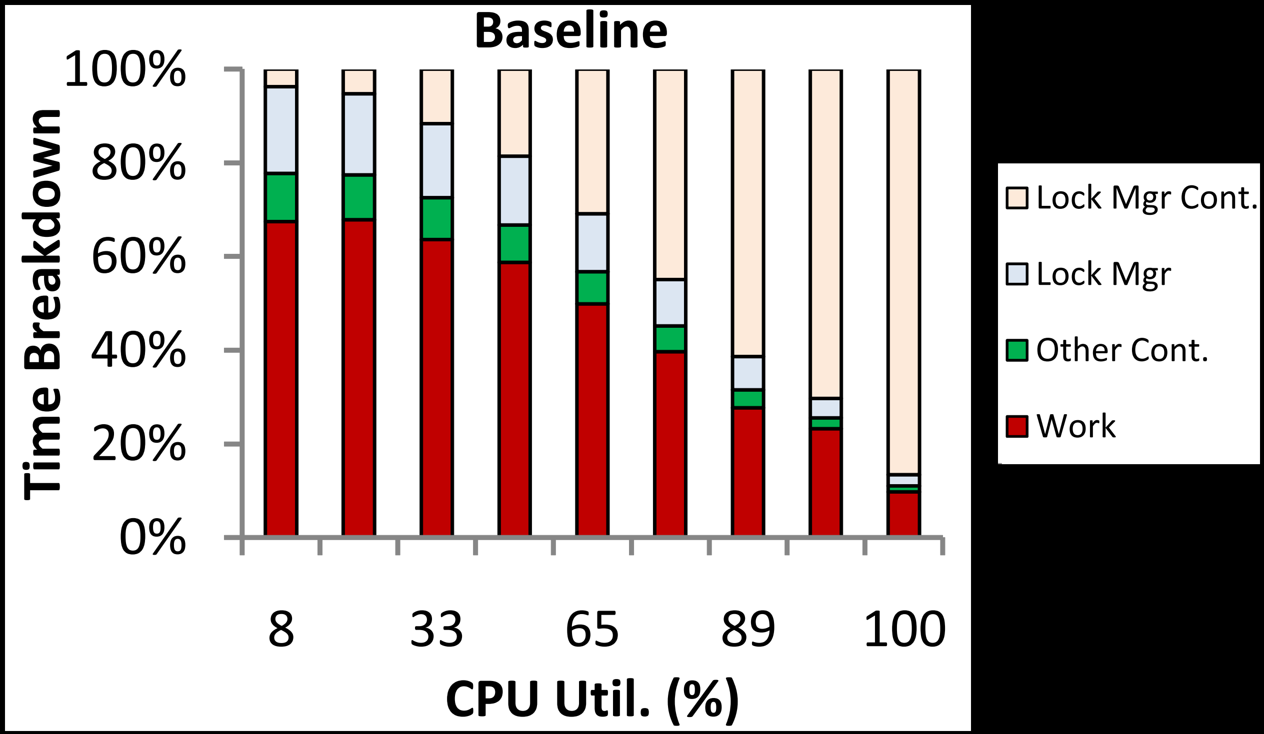
- Each thread only worries about its own transaction
  - no coordination among transactions
    - i.e. uncoordinated data access
  - leads to high lock contention, especially at data "hot spots"

Data Access Visualization



BROWN

Lock Contention As a Bottleneck



BROWN

The future looks bleak…

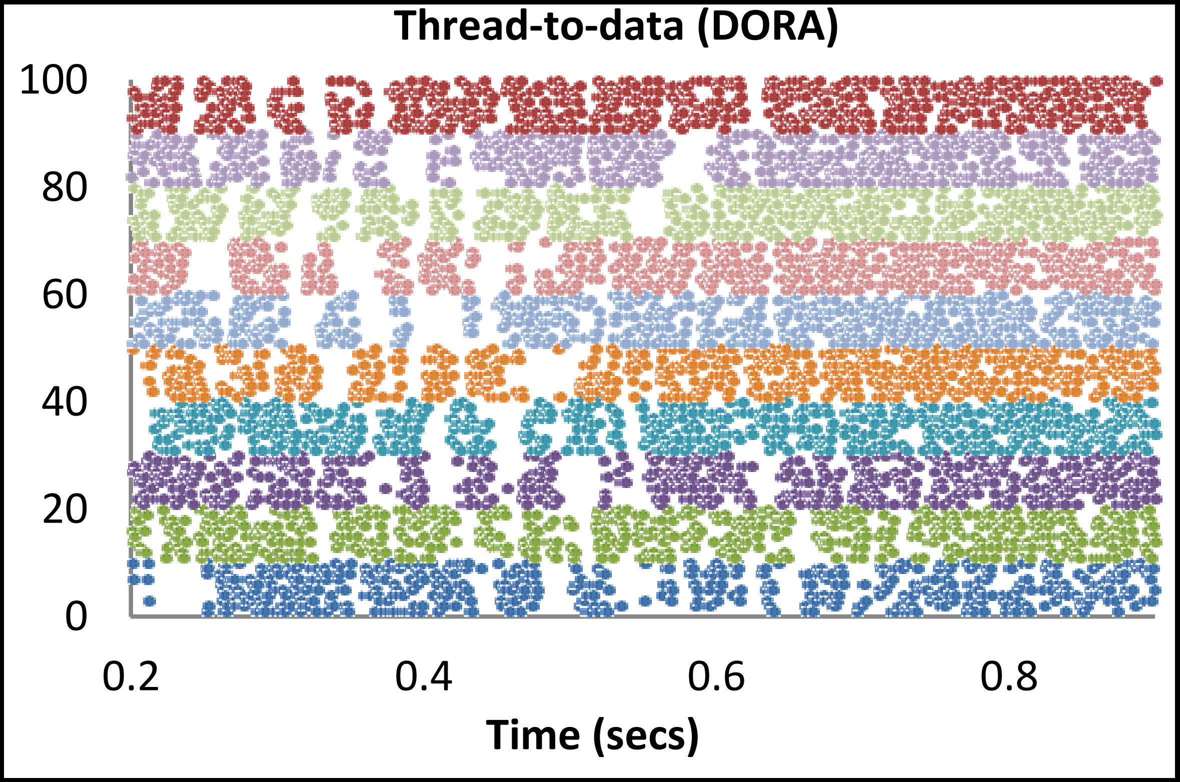
- Not quite!
- Idea: "Coordinate" data access patterns
  - rather than having threads contending for locks, have transactions contending for threads
  - distribute the transactions to the data, not data to the transactions

BROWN

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



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 executer (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

BROWN

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

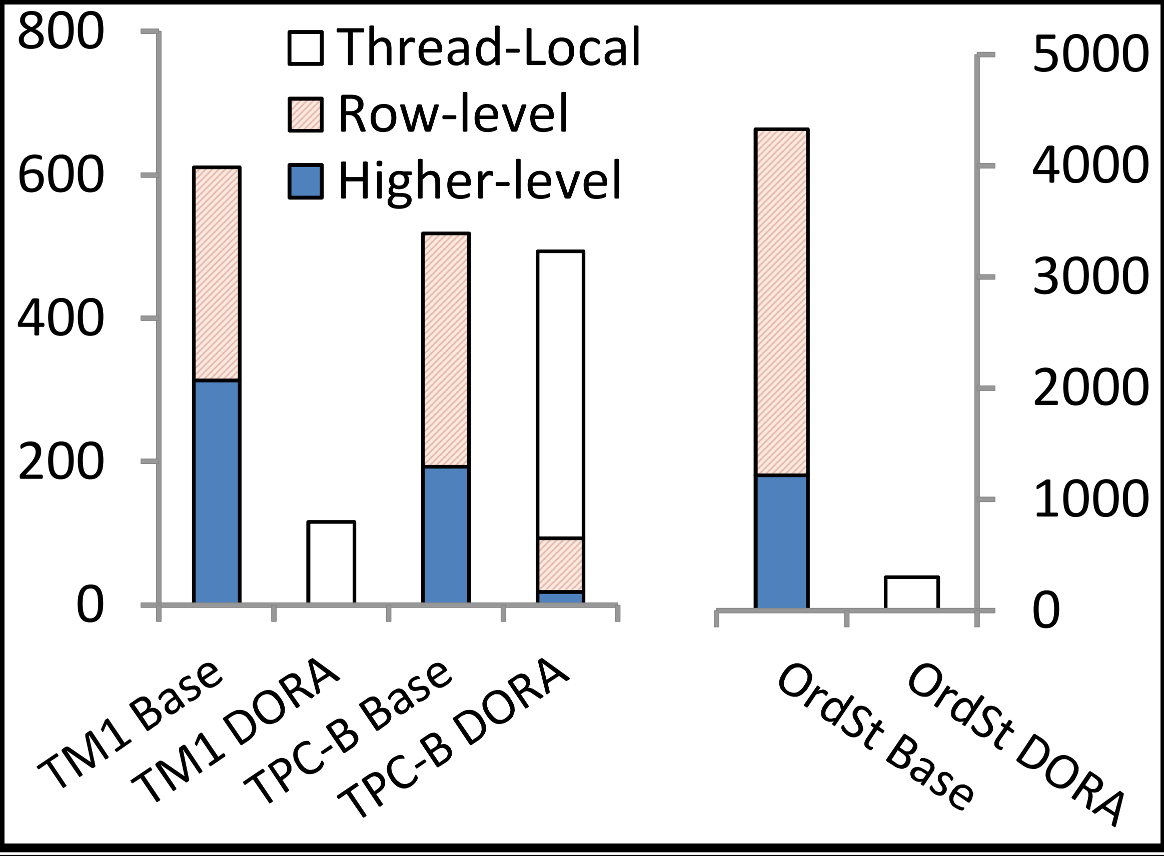
- Still need to acquire row-level locks through centralized locking manager
  - why?
    - T1 deletes a record
    - T2 inserts a record into the slot vacated by the record deleted by T1
    - T1 aborts but can’t roll back, slot is taken
  - row-level locks often not a source of contention

BROWN

Experimental Setup

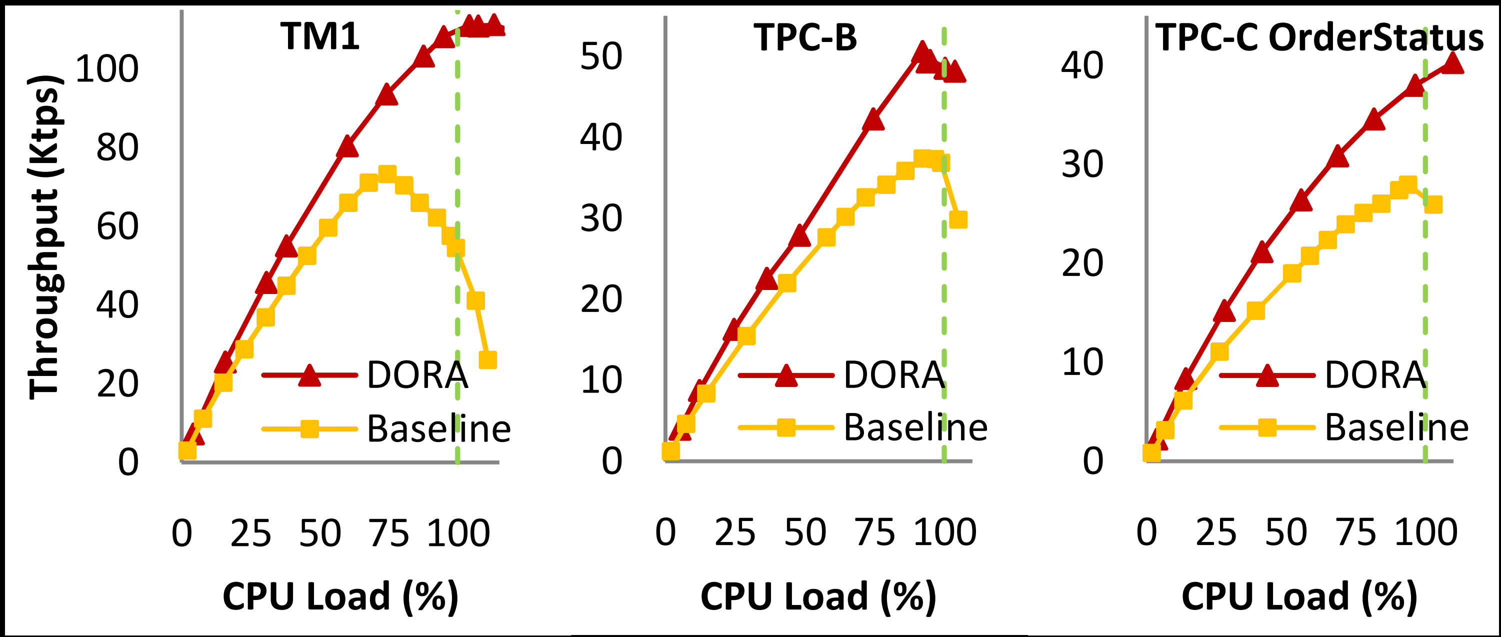
- 3 benchmarks used, all OLTP
  - TM-1
    - 7 transactions, 4 with updates
  - TPC-C
    - 150 warehouses (approx. 20 GB)
  - TPC-B
    - 100 branches (approx. 2 GB)

Lock Contention



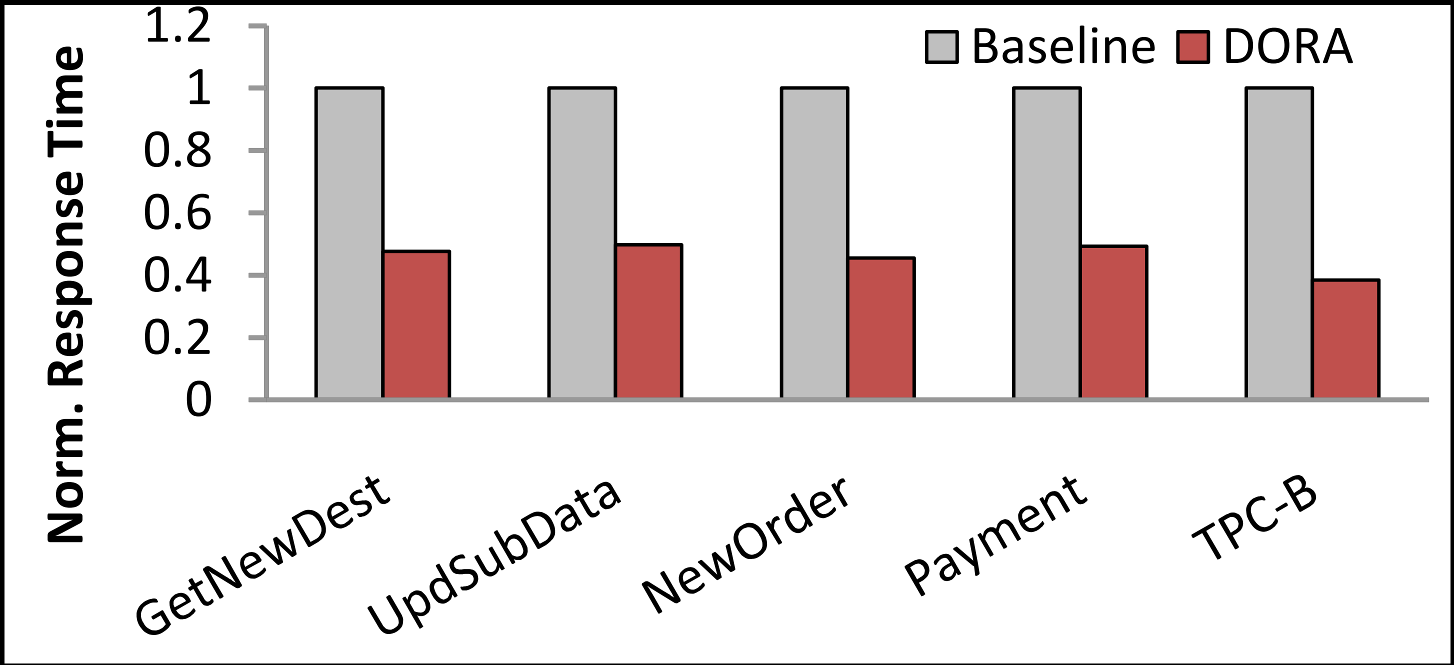
BROWN

Throughput



BROWN

Response Times



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

BROWN

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

BROWN

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

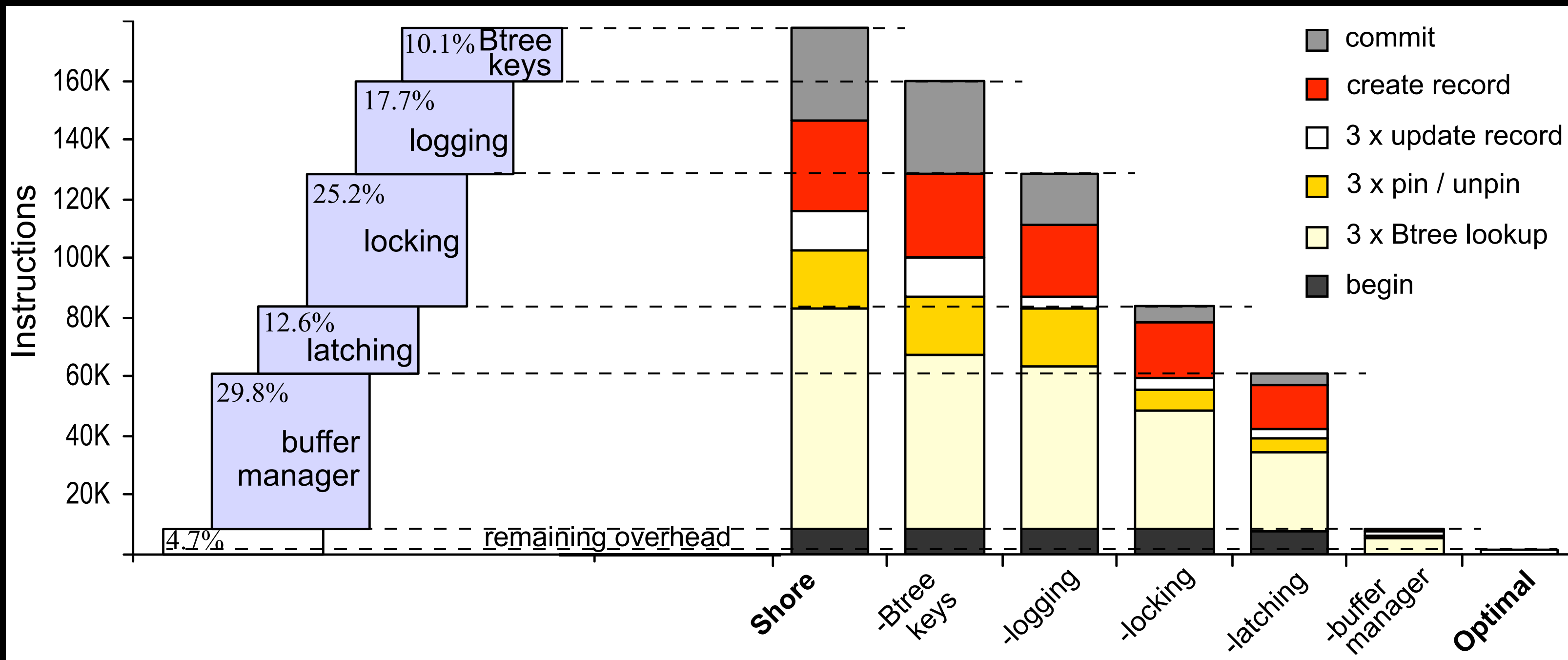


Figure 5. Detailed instruction count breakdown for Payment transaction.

BROWN

Instruction Count Breakdown

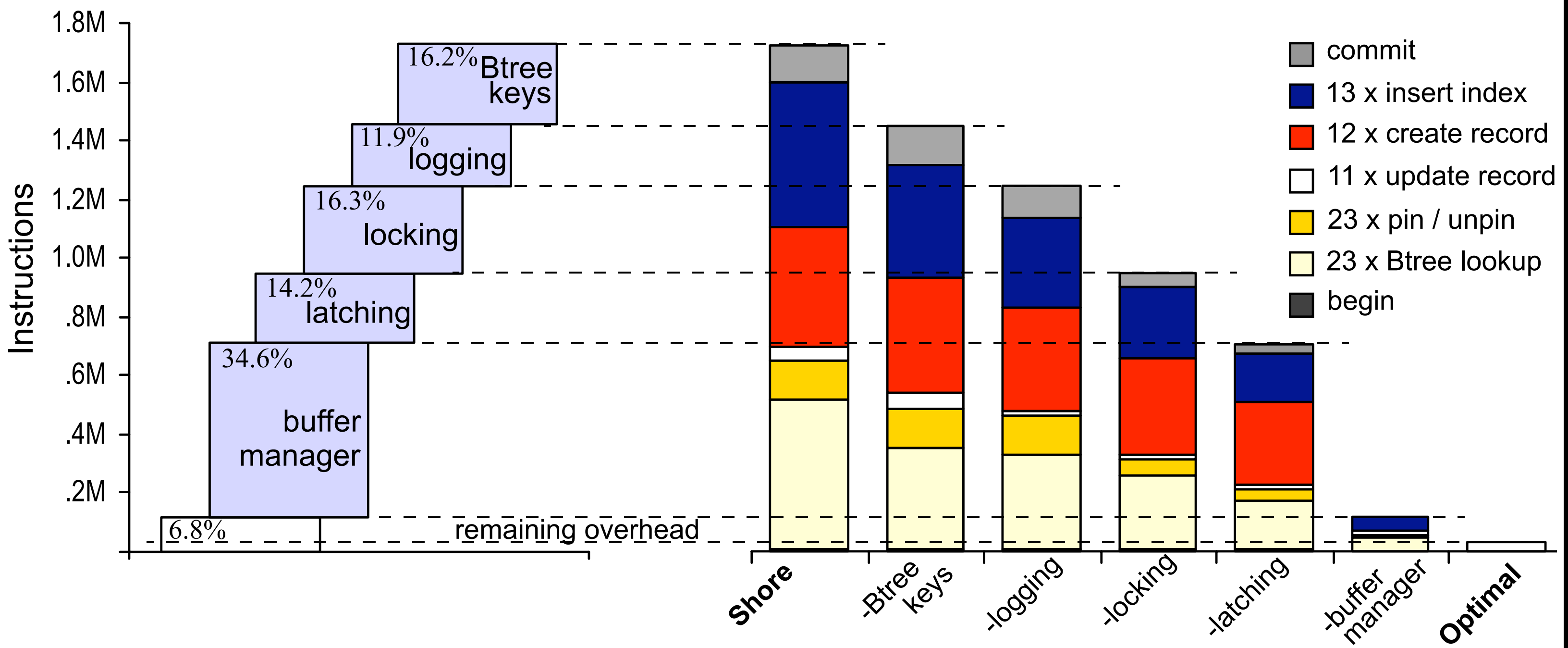


Figure 6. Detailed instruction count breakdown for New Order transaction.

BROWN

Conclusions

- Antiquated disk-based features can cause significant overhead in a main memory system
- Each component of a system should be carefully evaluated

BROWN

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

BROWN

Data Access Patterns

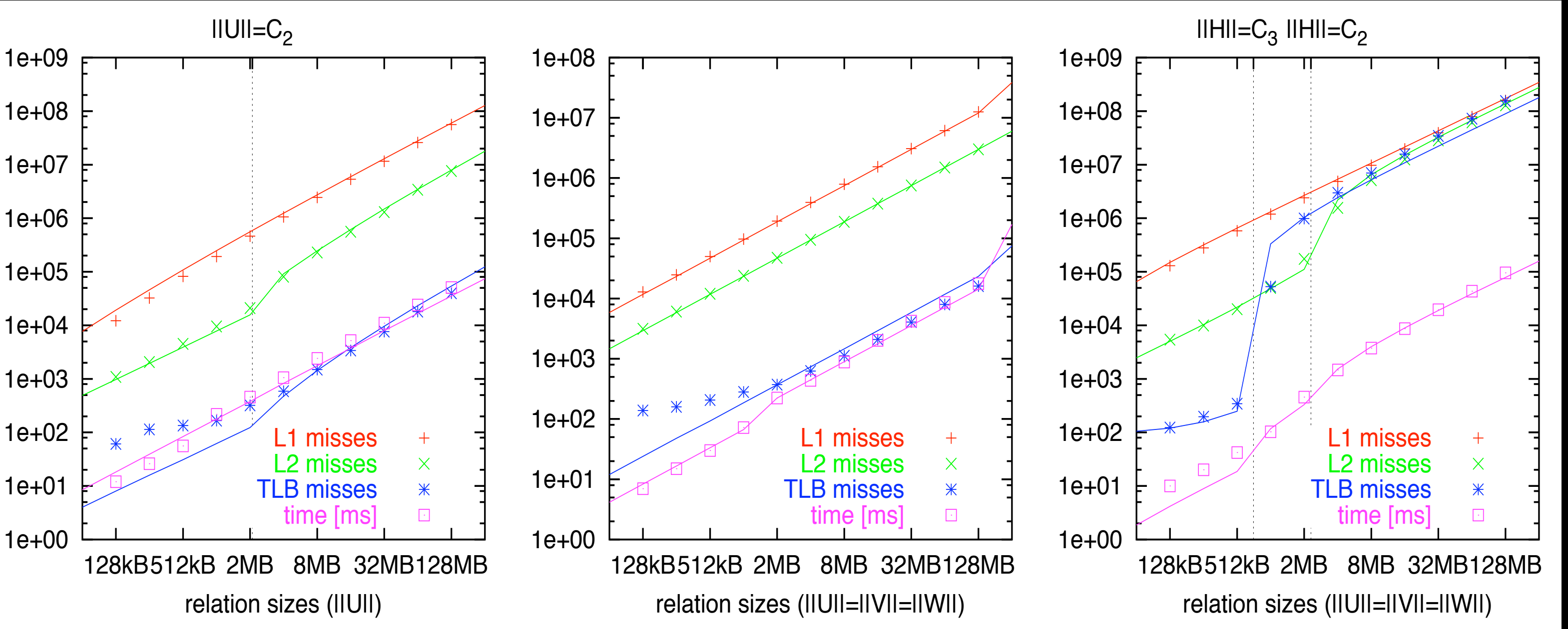
- different operators exhibit different data access patterns
  - pattern dictates both cost and number of caches misses
- How to accurately model access patterns?
  - basic access patterns
    - single/repetitive sequential traversal, single/repetitive random traversal, random
    - compound access patterns

BROWN

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



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

BROWN

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

BROWN

Questions?