The VoltDB Main Memory DBMS

Michael Stonebraker, Ariel Weisberg

3/19/19

Presented by: Sushant Raikar
Motivation

1990s
- Expensive main memory
- Slow processors

2010s
- Main memory is cheaper
- Faster processors

Future
- 100TBs can be deployed in memory
- PCM
Traditional DBMS

- 10% of time is spent on useful work
- 90% is just overhead
  - Buffer pool
  - Multi-threading
  - Locking
  - Logging

OLTP Through the Looking Glass, and What We Found There

Stavros Harizopoulos
HP Labs
Palo Alto, CA
stavros@hp.com

Daniel J. Abadi
Yale University
New Haven, CT
dna@cs.yale.edu

Samuel Madden
Massachusetts Institute of Technology
Cambridge, MA
(madden, stonebreaker)@csail.mit.edu

ABSTRACT

Online Transaction Processing (OLTP) databases include a suite of features — disk-resident B-trees and heap files, locking-based concurrency control, support for multi-threading — that were optimized for computer technology of the late 1970’s. Advances in modern processors, memories, and networks mean that today’s computers are vastly different from those of 30 years ago, such that many OLTP databases will now fit in main memory, and most OLTP transactions can be processed in milliseconds or less. Yet database architecture has changed little.

Based on this observation, we look at some interesting variants of conventional database systems that one might build that exploit recent hardware trends, and speculate on their performance through a detailed instruction-level break-down of the major components involved in a transaction processing database system (Shore) running a subset of TPC-C. Rather than simply profiling Shore, we progressively modified it so that after every feature removal or optimization, we had a (faster) working system that fully ran our workload. Overall, we identify overheads and optimizations that explain a total difference of about a factor of 20x in raw performance. We also show that there is no single “high holy grail” in a modern (memory resident) database system, but substantial time is spent in I/O, CPU, locking, B-
ASSUMPTIONS
Assumptions

- High Availability is necessary
- ACID is necessary
- Distribution of transactions
  - Single-node
  - One-shot
  - General
- Transactions are deterministic
Assumption: Cluster

- Clusters are deployed on LAN
  - Network partitions are rare
  - Better latency
- Replication over WAN
  - Asynchronous
ARCHITECTURE
DESIGN CHOICES
Buffer-pool

- Used for managing pages
- Tracks dirty or clean pages
- Not required for in-memory systems
  - Store data directly in memory
  - Use virtual memory pointers
Multi-threading & Locking

- Multi-threading causes race condition.
- VoltDB uses Share-nothing architecture
  - Memory chunk paired with single CPU
High Availability

- Data is partitioned
  - Consistent Hashing
- Each partition is replicated
  - K-factor

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>1.50</td>
</tr>
<tr>
<td>3</td>
<td>Charlie</td>
<td>30</td>
</tr>
</tbody>
</table>
Consistency

- Deterministic Transaction [Assumption]
- Uses “single partition initiator” to order transactions
- Reads are added without SPI
Consistency (Multi-node)

- Queries that span multiple partitions
  - Go through Multi-Partition initiator
  - C -> M -> B
Persistence

- Global Errors
- Solution:
  - Snapshot
  - Logging
Persistence [Snapshot]

A

SPI #1

PARTITION #1 SERVER 1
PARTITION #1 SERVER 2

A
A

Executor

D

snapshot

A
C

snapshot

B

A

snapshot

A
Persistence [Logging]
Development Focus in 2013

- SQL
- On-Line Reprovisioning
- Downstream Repositories
Application Areas

- High Speed updates
- Maintaining a state
  - Internet Games, Leaderboards
  - IOT
  - High Frequency Trading
Performance

- Voter Benchmark
  - Obeyed assumptions
  - Workload
    - Vote a candidate
    - Produce heat map for all states
    - Produce leaderboard once a second
  - Linear Scalability
    - 5-node system -> 0.6 million transactions/second
    - 30 node system -> 3.4 million transactions/second
SUMMARY
Summary

- Traditional Design on main memory can do 10% of useful work
- No Buffer Pool
- No Multithreading, hence no locking
- Deterministic ordering
- Logging and Snapshot
“One way to deal with a problem is to not have it.”

---

VoltDB
THANK YOU
Summary

- TPC-Benchmark, indicates 10% of useful work
- No Buffer Pool
- No Multithreading, hence no locking
- Deterministic ordering
- Logging and Snapshot
Discussion Points

- How would you enforce deterministic transactions?
- What will be the impact of non-volatile memory on VoltDB?
- What could’ve been a better performance evaluation?
- Can group-commits cause inconsistency?
- Which point-of-view do you agree on, LeanStore (Buffer pool) VS VoltDB (No Buffer pool)?