NV-RAM-aware Logging in Transaction Systems

Jian Huang

Karsten Schwan  Moinuddin K. Qureshi

Georgia Tech  cercs  CARET
Logging Support for Transactions

ATOMICITY  CONSISTENCY  ISOLATION  DURABILITY
Logging Support for Transactions

ARIES: Disk-Based Approach (TODS’92)

Write-ahead Logging
Logging Support for Transactions

ARIES: Disk-Based Approach (TODS’92)

- Write-ahead Logging
- Rollback
Logging Support for Transactions

**ACID**
- ATOMICITY
- CONSISTENCY
- ISOLATION
- DURABILITY

**ARIES: Disk-Based Approach (TODS’92)**

- Write-ahead Logging
- Rollback
- Centralized Log Buffer
On the Logging Persistence

TXN_A:
- lock
- commit (flushing log)
- unlock

TXN_B:
- acquire lock
- wait for lock
- commit (flushing log)
- unlock
On the Logging Persistence

Sync Commit

Disk I/O Is the Bottleneck!
On the Logging Persistence

Sync Commit

TXN_A:
- lock
- commit (flushing log)
- unlock

TXN_B:
- acquire lock
- wait for lock
- commit (flushing log)  
- unlock

Async Commit

TXN_A:
- lock
- unlock
- commit (flushing log)

TXN_B:
- lock
- unlock
- commit (flushing log)
On the Logging Persistence

**Sync Commit**

TXN_A:
- lock
- commit (flushing log)
- unlock

TXN_B:
- acquire lock
- wait for lock
- commit (flushing log)
- unlock

**Disk I/O Is the Bottleneck!**

**Async Commit**

TXN_A:
- lock
- unlock
- commit (flushing log)

TXN_B:
- lock
- unlock
- commit (flushing log)

At the risk of partial data loss and incorrect result!
Rethink the Logging with NVRAM

Close to DRAM
Rethink the Logging with NVRAM

Close to DRAM + Byte Addressability
Rethink the Logging with NVRAM

- Close to DRAM
- Byte Addressability
- Non-Volatility
Rethink the Logging with NVRAM

Disk-based logging

- Avoid frequent disk access
- Obtain sequential access pattern
Rethink the Logging with NVRAM

**Flash-based logging**

- Reduce log flushing time
- Atomic Write (OSDI’08)

**Disk-based logging**

- Avoid frequent disk access
- Obtain sequential access pattern

**Benefits of NVRAM**

- Close to DRAM
- Byte Addressability
- Non-Volatility
Rethink the Logging with NVRAM

Disk-based logging
- Avoid frequent disk access
- Obtain sequential access pattern

Flash-based logging
- Reduce log flushing time
- Atomic Write (OSDI’08)

NVRAM-based

Close to DRAM
Byte Addressability
Non-Volatility
How to Use NVRAM in DB System?

Traditional Design

Transactions

- Log Buffer (DRAM)
- Page Cache (DRAM)
- Log and DB Tables (HDD/SSD)
How to Use NVRAM in DB System?

Traditional Design

Transactions

- Log Buffer (DRAM)
- Page Cache (DRAM)
- Log and DB Tables (HDD/SSD)

All-in-NVRAM

Transactions

- Log Buffer (DRAM)
- Page Cache (DRAM)
- Log and DB Tables (NVRAM)
How to Use NVRAM in DB System?

Traditional Design

Transactions

Log Buffer (DRAM)  Page Cache (DRAM)

Log and DB Tables (HDD/SSD)

All-in-NVRAM

Transactions

Log Buffer (DRAM)  Page Cache (DRAM)

Log and DB Tables (NVRAM)

NVLogging
How to Use NVRAM in DB System?

Traditional Design

<table>
<thead>
<tr>
<th>Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Buffer (DRAM)</td>
</tr>
<tr>
<td>Page Cache (DRAM)</td>
</tr>
<tr>
<td>Log and DB Tables (HDD/SSD)</td>
</tr>
</tbody>
</table>

All-in-NVRAM

<table>
<thead>
<tr>
<th>Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Buffer (DRAM)</td>
</tr>
<tr>
<td>Page Cache (DRAM)</td>
</tr>
<tr>
<td>Log and DB Tables (NVRAM)</td>
</tr>
</tbody>
</table>

Which one we should take?

<table>
<thead>
<tr>
<th>Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Buffer (NVRAM)</td>
</tr>
<tr>
<td>Page Cache (DRAM)</td>
</tr>
<tr>
<td>DB Tables (HDD/SSD)</td>
</tr>
</tbody>
</table>

NVLogging
Using NVRAM in Cost-Effective Way
Using NVRAM in Cost-Effective Way

- all-in-HDD
- all-in-SSD

Diagram displaying normalized TPS (all-in-HDD as the baseline) for TPCC, TATP, and TPCB.
Using NVRAM in Cost-Effective Way

- all-in-HDD
- all-in-SSD
- all-in-NVRAM

Normalized TPS (all-in-HDD as the baseline)

- TPCC
- TATP
- TPCB

51x

Using NVRAM in Cost-Effective Way

- tpcc
- tatp
- tpcb

$\$\$\$$

Using NVRAM in Cost-Effective Way

- tpcc
- tatp
- tpcb

$\$\$\$$

Using NVRAM in Cost-Effective Way

- tpcc
- tatp
- tpcb

$\$\$\$$

Using NVRAM in Cost-Effective Way

- tpcc
- tatp
- tpcb

$\$\$\$$
Using NVRAM in Cost-Effective Way

- **all-in-HDD**
- **all-in-SSD**
- **all-in-NVRAM**
- **db-in-SSD, log-in-NVRAM**

![Bar Chart]

<table>
<thead>
<tr>
<th></th>
<th>Normalized TPS (all-in-HDD as the baseline)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TPCC</strong></td>
<td>![Bar Chart]</td>
</tr>
<tr>
<td><strong>TATP</strong></td>
<td>![Bar Chart]</td>
</tr>
<tr>
<td><strong>TPCB</strong></td>
<td>![Bar Chart]</td>
</tr>
</tbody>
</table>

- $18\%$ cost saving using NVRAM in TATP
Using NVRAM in Cost-Effective Way

- **all-in-HDD**
- **all-in-SSD**
- **all-in-NVRAM**
- **db-in-HDD, log-in-NVRAM**
- **db-in-SSD, log-in-NVRAM**

### Normalized TPS (all-in-HDD as the baseline)

<table>
<thead>
<tr>
<th>Method</th>
<th>TPCC</th>
<th>TATP</th>
<th>TPCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>all-in-HDD</td>
<td>20</td>
<td>25</td>
<td>12</td>
</tr>
<tr>
<td>all-in-SSD</td>
<td>25</td>
<td>30</td>
<td>15</td>
</tr>
<tr>
<td>all-in-NVRAM</td>
<td>30</td>
<td>35</td>
<td>20</td>
</tr>
<tr>
<td>db-in-HDD, log-in-NVRAM</td>
<td>35</td>
<td>40</td>
<td>25</td>
</tr>
<tr>
<td>db-in-SSD, log-in-NVRAM</td>
<td>40</td>
<td>45</td>
<td>30</td>
</tr>
</tbody>
</table>

- 18% improvement in cost
- 26% improvement in cost
Using NVRAM in Cost-Effective Way

- all-in-HDD
- all-in-NVRAM
- db-in-HDD, log-in-NVRAM
- all-in-SSD
- db-in-SSD, log-in-NVRAM

Normalized TPS (all-in-HDD as the baseline)

<table>
<thead>
<tr>
<th></th>
<th>TPCC</th>
<th>TATP</th>
<th>TPCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>all-in-HDD</td>
<td>25</td>
<td>18%</td>
<td>26%</td>
</tr>
<tr>
<td>all-in-NVRAM</td>
<td>30</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>db-in-HDD, log-in-NVRAM</td>
<td>35</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>all-in-SSD</td>
<td>40</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>db-in-SSD, log-in-NVRAM</td>
<td>45</td>
<td>$</td>
<td>$</td>
</tr>
</tbody>
</table>
Using NVRAM in Cost-Effective Way

- all-in-HDD
- all-in-NVRAM
- db-in-HDD, log-in-NVRAM
- all-in-SSD
- db-in-SSD, log-in-NVRAM

Bridging the performance gap with less cost
Bottlenecks Shifted from I/O to Software

- log-in-HDD, db-in-NVRAM
- log-in-SSD, db-in-NVRAM
- log-in-NVRAM, db-in-HDD
- log-in-NVRAM, db-in-SSD
- all-in-NVRAM

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Misc</th>
<th>DB Operations</th>
<th>Lock Manager</th>
<th>Log Contention</th>
<th>Log Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bottlenecks Shifted from I/O to Software

- log-in-HDD, db-in-NVRAM
- log-in-SSD, db-in-NVRAM
- log-in-NVRAM, db-in-HDD
- log-in-NVRAM, db-in-SSD
- all-in-NVRAM

- Misc
- DB Operations
- Lock Manager
- Log Contention
- Log Operations
Bottlenecks Shifted from I/O to Software

Software-induced overhead becomes the new bottleneck!
NVLogging: Redesign the Logging with NVRAM

1. Use NVRAM in Cost-Effective Way
2. Avoid the Software Overhead
3. Minimal Code Modifications
Log Management in NVLogging
Log Management in NVLogging
Log Management in NVLogging
Log Management in NVLogging

- Log_head
- Log_tail
- Log Entry
- States: 0: DRAM, 1: NVRAM
- Pointer
- LogRec object

DRAM

NVRAM
Log Management in NVLogging

- LogRec
- Log_head
- States
- Pointer
- Log_entry
- Log_tail
- 0: DRAM
- 1: NVRAM
- LogRec object
- DRAM
- NVRAM
Log Management in NVLogging

Processor cache and DRAM are volatile.

LogRec

Log_head

States

0: DRAM

1: NVRAM

Log_tail

Log Entry

LogRec object
Log Management in NVLogging

Log persistence:
flush-on-insert & flush-on-commit

Log Entry

States

Pointer

0: DRAM
1: NVRAM

LogRec object

LogRec

Log_head

Log_tail

Log Entry Index

DRAM

NVRAM
Log Management in NVLogging

Log persistence:
flush-on-insert & flush-on-commit

DRAM
NVRAM

Log Rec
Log Entry
Atomic Update

Log Entry
Index

Log_head
States
0: DRAM
1: NVRAM

Log_tail
Pointer
Avoid Log Contention with Per-Transaction Logging
Avoid Log Contention with Per-Transaction Logging

TX_A

LSN #0
LogRec

LSN #2
LogRec

LSN #3
LogRec

TX_B

LSN #1
LogRec

LSN #4
LogRec

DRAM

NVRAM

Log Entry

Index

States

0: DRAM
1: NVRAM

Pointer
Avoid Log Contention with Per-Transaction Logging

No Cross-space Pointer

States | Pointer
---|---
0: DRAM
1: NVRAM
Avoid Log Contention with Per-Transaction Logging

Using back pointer to access log objects

Log Entry Index

Recovery
Avoid Log Contention with Per-Transaction Logging

TX_A

LSN #0
LogRec

LSN #2
LogRec

LSN #3
LogRec

TX_B

LSN #1
LogRec

LSN #4
LogRec

Log Entry Index

Log Truncation

DRAM

NVRAM
Implementation and Experimental Setup

Shore-MT + Shore-Kits Benchmark

Transactions

- NVLogging
- NVRAM

Page Cache (DRAM)
- DB Tables

HDD/SSD
Implementation and Experimental Setup

**Shore-MT** + Shore-Kits Benchmark

![Diagram showing the implementation setup with NVLogging, Page Cache (DRAM), DB Tables, NVRAM, and HDD/SSD]

Intel Persistent Memory Emulator Platform

**Xeon-EP @ 2.6 GHz, 64 GB DRAM + 256 GB NVRAM**
Benefits on Throughput with TPCB

Throughput (K TPS)

- db-in-HDD, log-in-NVRAM
- db-in-SSD, log-in-NVRAM
- Distributed Logging
- NVLogging

NVRAM Latency (nanoseconds)

- DRAM
- 300
- 400
- 500

Throughput (K TPS) vs. NVRAM Latency (nanoseconds)
Benefits on Throughput with TPCB

- db-in-HDD, log-in-NVRAM
- db-in-SSD, log-in-NVRAM
- Distributed Logging
- NVLogging

Throughput (K TPS)

<table>
<thead>
<tr>
<th>NVRAM Latency (nanoseconds)</th>
<th>DRAM</th>
<th>300</th>
<th>400</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td>400</td>
<td>55</td>
<td>65</td>
<td>75</td>
<td>85</td>
</tr>
<tr>
<td>500</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
</tr>
</tbody>
</table>

Throughput improvement: 83%
Benefits on Throughput with TPCB

NVLogging improves TPS by 2.3x, compared to the baseline (db-in-SSD, log-in-NVRAM)
Benefits on Cost Savings

Cost Ratio (\frac{\text{\$NVRAM}}{\text{\$Disk}})

Normalized TPS$/

- **all-in-NVRAM**
- **Distributed Logging**
- **NVLogging**

1. Benefits on Cost Savings

\[ \text{Cost Ratio} = \frac{\text{\$NVRAM}}{\text{\$Disk}} \]
Benefits on Cost Savings

Cost Ratio (\(\frac{\text{NVRAM}}{\text{Disk}}\))

Normalized TPS/$
Benefits on Cost Savings

Normalized TPS/$ vs Cost Ratio ($NVRAM / $Disk) for all-in-NVRAM, Distributed Logging, and NVLogging.
Cost-effective Solution for DB system
Higher TPS/$

Per-transaction Logging with NVRAM
Exploit NVRAM’s both non-volatility and byte-addressability
Thanks!

Jian Huang
jian.huang@gatech.edu

Karsten Schwan    Moinuddin K. Qureshi

Q&A