Adaptive Recovery for SCM-Enabled Databases
Ismail Oukid (TU Dresden & SAP), Daniel Bossle* (SAP), Anisoara Nica (SAP), Peter Bumbulis (SAP), Wolfgang Lehner (TU Dresden), Thomas Willhalm (Intel)

* Contributed while interning at SAP

ADMS@VLDB, September 1st, 2017
Non-Volatile Memory

We assume hardware-based wear-leveling

Limited write endurance

Low, asymmetric latency

Denser than DRAM

Energy efficient

Byte-addressable

NVM

Writes noticeably slower than reads

More capacity and cheaper than DRAM → 3 TB per socket for first-gen 3D XPoint

NVM is a merging point between main memory and storage
Architecting NVM

- An NVM-optimized filesystem provides zero-copy mmap
- Direct access to NVM via load/store instructions
- Several filesystem proposals: NOVA, PMFS, SCMFS, etc.
- Linux ext4 and xfs already provide Direct Access support

NVM may become a universal memory
SOFORT: A Hybrid NVM-DRAM Storage Engine

→ Primary data persisted in and accessed from NVM
→ Secondary data can be persistent, transient, or hybrid

NVM enables a single-level database storage architecture
Recovery Time

Rebuilding secondary data is the new recovery bottleneck

Goal: improve recovery without compromising query performance
Synchronous Recovery

Recovery protocol
1. Recover primary data
2. Undo in-flight TXs
3. Rebuild secondary data
4. Accept queries

+ Secondary data rebuilt as fast as possible
- System is not responsive during recovery
Asynchronous Recovery (aka Instant Recovery)

**Intuition:** Primary data is sufficient to answer queries

Accept queries right after recovering primary data

During recovery
- Dictionary index lookup replaced with dictionary array scan
- Column index lookups replaced by column scans
- CPU resources split between query processing and recovery

+ Near-instant responsiveness of the database
- It takes longer to reach pre-failure throughput
Adaptive Recovery

**Intuition 1**: Secondary data structures are not equally important

Recover indexes in the order of their importance

**Intuition 2**: Some secondary data structures are not useful for the currently running workload

Release recovery resources after rebuilding important indexes

How to decide the importance of secondary data structures?
Index Benefit Functions

**Benefit\_indep**

Computes the benefit of an index for a query plan independently of other indexes

\[
\text{Benefit}\_\text{indep}(s, Q, S) = \text{Cost}(Q, S_{\text{NVM}}) - \text{Cost}(Q, S_{\text{NVM}} \cup \{s\})
\]

- **S**: Set of all indexes
- **s**: Considered index
- **Q**: Considered query
- **S\_NVM**: Set of persistent indexes

**Benefit\_dep**

Computes the benefit of an index while captures its dependencies to currently available indexes

\[
\text{Benefit}\_\text{dep}(s, Q, S) = \text{Cost}(Q, S(t_Q)) - \text{Cost}(Q, S(t_Q) \cup \{s\})
\]

- **S(t_Q)**: Set of available indexes at time \(t_Q\)
Ranking of Secondary Data Structures

WoPast and WoRestart can be similar or different → A ranking function must take both into consideration

\[ \text{rank}(s,t) = \alpha(n) \times \text{Benefit}(s, \text{WoPast}, S) + (1 - \alpha(n)) \times \text{Benefit}(s, \text{WoRestart}(t), S) - \text{rebuild}(s) \]

e.g., \( \alpha(n) = \alpha^n \) and \( 0 < \alpha < 1 \) → WoPast’s weight decays with the number of statements in WoRestart
Evaluation Setup

- Intel NVM Emulator
  → Intel Xeon E5 @2.60Ghz, 20MB L3 cache, 8 physical cores
- Benchmarks run on a single socket
### Experimental Setup (Cont’d)

**TATP Benchmark**
- 80% Get Subscriber Data
- 20% Update Location

**TPC-C Benchmark**
- 50% Order Status
- 50% Stock Level

**Sofort Configuration**
- 8 users (threads)
- All indexes in DRAM

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Query Processing Resources</th>
<th>Recovery Resources</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>$rk.Qx.Rz$</td>
<td>static $x$ cores</td>
<td>static $z$ cores</td>
<td>yes</td>
</tr>
<tr>
<td>$\neg rk.Qx.Rz$</td>
<td>static $x$ cores</td>
<td>static $z$ cores</td>
<td>no</td>
</tr>
<tr>
<td>$rk.Q^{ad}.Rz^{ad}$</td>
<td>adaptive</td>
<td>adaptive, start with $z$ cores</td>
<td>yes</td>
</tr>
</tbody>
</table>
Recovery Strategies

Recovery workload same as pre-failure workload

Asynchronous recovery worse than synchronous recovery!

Pre-failure throughput regained before the end of recovery
Workload Change During Recovery

Pre-failure workload: Full TATP and TPC-C mixes

Recovery workload: only TATP and TPC-C queries presented earlier

Adaptive recovery adapts well to workload changes
Worst-Case Analysis

Synthetic benchmark
- 10 tables, 10 columns and 1 Mio row each
- All columns uniformly queried

Resources are released as soon as the job queue is empty

Synchronous recovery is a lower bound for adaptive recovery

<table>
<thead>
<tr>
<th>Legend Entry</th>
<th>Rec. End</th>
<th>#TXs t=17s</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \neg rk.Q0.R8 )</td>
<td>16.6 s</td>
<td>0.29 M</td>
</tr>
<tr>
<td>( rk.Q^{ad}.R8^{ad} )</td>
<td>16.6 s</td>
<td>0.84 M</td>
</tr>
</tbody>
</table>
Conclusion

➢ NVM enables a single-level database storage architecture

➢ Rebuilding secondary data is the new recovery bottleneck

➢ Regaining pre-failure throughput near-instantly possible, but at a significant query performance cost

➢ When query performance is paramount, adaptive recovery allows to swiftly regains pre-failure throughput
Thank you.

Contact information:

Ismail Oukid
ismail.oukid@sap.com
Resource Allocation

Recovery workload same as pre-failure workload

Only a subset of indexes is relevant to the workload

<table>
<thead>
<tr>
<th>Legend Entry</th>
<th>TATP</th>
<th>TPCC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rec. End</td>
<td>#TXs t=15 s</td>
</tr>
<tr>
<td>( r_k.Q^0.R8 )</td>
<td>11.6 s</td>
<td>4.3 M</td>
</tr>
<tr>
<td>( r_k.Q^a^d.R8^a^d )</td>
<td>63.2 s</td>
<td>6.1 M</td>
</tr>
</tbody>
</table>

Adapting resources significantly improves recovery performance