FOEDUS: OLTP Engine for a Thousand Cores and NVRAM

Hideaki Kimura
HP Labs, Palo Alto, CA

Slides By : Hideaki Kimura
Presented By : Aman Preet Singh
Next-Generation Server Hardware?

Differences

- HP Photonics? Infiniband? QPI?
- HP Memristor? PCM? Flash?
- ...

Commonalities

- 1,000s of CPU Cores
- Fast Interconnect
- Huge Low-Latency NVRAM
- Endurance will be an issue
GAP?

- Traditional OLTP databases do not scale to large number of cores.
- Recent main-memory databases achieve better scalability but they do not efficiently handle NVRAM.
GAP?

• Software should place data wisely.
  • System on a chip.
  • Systems capable of handling cache in-coherrent architectures.
  • Non-uniform Memory access (NUMA) aware systems.

• Aim should be to try to make the data local.
Solution?

- To provide these capabilities, FOEDUS was built.
  - Ground up database
  - Open source
  - Fully ACID
  - Designed to scale up to thousands of cores
  - Makes the best use of DRAM for OLTP,
  - And NVRAM for OLAP queries.
FOEDUS Key Principles

• Bring in-memory speed to NVRAM
  • SILO-like Lightweight Optimistic Concurrency Control

• Dual-Page: physically independent, logically equivalent
  • Mutable Volatile Pages in DRAM
  • Immutable Snapshot Pages in NVRAM

• Master-Tree: Simple and Scalable OCC for NVRAM
  • Masstree + Foster B-Tree + Foster-Twin
SILO: Lightweight/Decentralized OCC [Tu et al]

Read Version
↓
Fence
↓
Read Data
↓
Fence
↓
Read Version

Record
Epoch
Status-bits
Version (64 bits)
Data (Payload)

Commit Time

Writer Side (After Commit)
Write Data
↓
Fence
↓
Write Epoch
(==unlock)

Same Version?
↓
yes, commit
↓
no, abort (retry).

(consume or acquire)
(acquire)
SILO: Decentralized Logger with Epoch [Tu et al]

For each epoch (10ms~50ms)

Durable Committed Tail

Disk

Log Writer

Log Manager

SOC

Epoch File
- Global Durable Epoch
- Offsets etc

© Copyright 2015 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice.
SILO’s OCC Benefits

✓ **Extremely Lightweight and Scalable**
Block-free (lock-free with retries)
No Write for Reads (cf., read-lock)
✓ **In-page Lock Mechanism**
No centralized lock manager
Cache Friendly

*But, how can we go beyond DRAM?*
In-Memory DBMS Beyond DRAM

a) Traditional Databases

b) H-Store/Anti-Cache [Pavlo et al.]

c) Hekaton/Siberia [Larson et al.]

d) FOEDUS

No Logical Equivalence

Physically Dependent

Volatile pages

Snapshot pages

Dual Pages
Logically Equivalent, Physically Independent Dual

- **Volatile Page**: Mutable, on DRAM
- **Snapshot Page**: Immutable, on NVRAM

<table>
<thead>
<tr>
<th>Volatile Pointer</th>
<th>Snapshot Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>X</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>Y</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
</tr>
</tbody>
</table>

- Drop Volatile (if X≡Y)
- :)nullptr
- :Volatile-Page is just made. No Snapshot yet.
- :Snapshot-Page is latest. No modification.
- :X is the latest truth, but Y may be equivalent.
Constructing Stratified Snapshot from Logical Logs

1. Assign Partitions

2. Run Mappers/Reducers

3. Combine Root/Metadata

4. Install/Drop Pointers

Batch-Apply

New Snapshot
Benefits of Stratified Snapshots

- Serializable transactions check only a single version of data ↔ LSM-Trees
- Drastically more scalable and efficient construction of NVRAM-resident data pages
  - Separate from transactions/logging
  - Everything Batched, Processed in a tight loop
  - Large Sequential Write only. No frequent flush.
Master-Tree in a Nutshell

- **Mass-tree:** *Cache Crafty B-tree* and *OCC* with *in-page* Lock Mechanism.
- **Foster B-tree:** *Single incoming pointer via foster-child* to ease page in/out.
- **Foster-Twins:** Drastically Simplifies OCC and Reduces Aborts/Retries.
OCC Problem 1: Unnecessary Retries/Aborts

Further, SILO must take Page-Version set and abort if changed by pre-commit.

Lots of Aborts!
Foster Twins here to Save You!

• Strong Invariants to drastically simplify OCC and eliminate unnecessary aborts/retries
• Still everything in-page
  • No per-tuple GC or compaction/migration
  • No centralized data structure
**Mass-tree**

- Foster B-Tree
- Bw-Tree

**Master-Tree**

- All data/metadata placed **in-page**
- **Immutable range**
- All information **always track-able**
- All invariants **recursively** hold

---

© Copyright 2015 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice.
Algorithm 1: SILO precommit protocol [13]

Input: R: Read set, W: Write set, N: Node set

/* Precommit-lock-phase */
Sort W by unique order;
foreach w ∈ W do Lock w;
Fences, get commit epoch;
/* Precommit-verify-phase */
foreach r, observed ∈ R do if r.tid ≠ observed and r ∉ W then abort;
foreach n, observed ∈ N do if n.version ≠ observed then abort;
Generate TID, apply W, and publish log;

Algorithm 2: FOEDUS precommit protocol

Input: R: Read set, W: Write set, P: Pointer set

/* Precommit-lock-phase */
while until all locks are acquired do
    foreach w ∈ W do if w.tid.is-moved() then w.tid ← track-moved(w.page, w.record)
        Sort W by unique order;
        foreach w ∈ W do Try lock w. If we fail and find that w.tid.is-moved(), release all locks and retry
end
Fences, get commit epoch;
/* Precommit-verify-phase */
foreach r, observed ∈ R do
    if r.tid.is-moved() then r.tid ← track-moved(r.page, r.record)
    if r.tid ≠ observed and r ∉ W then abort;
foreach p ∈ P do if p.volatile-ptr ≠ null then abort;
Generate TID, apply W, and publish log;
Dual Pages: Benefits

✓ **Snapshot Pages are Immutable**
Drastically simplifies Caching/Replication/Traversal/Commit/etc

✓ **Physically Independent**
Transactions never interfere w/ construction/retrieval/eviction of Snapshot Pages

✓ **Logically Equivalent**
- No Bloom Filter or global data needed for Serializability ↔ LSM-Trees
- Volatile Pages guaranteed to contain latest data
- Snapshot Pages guaranteed to be complete as of snapshot-epoch
Experiments

- **TPC-C, Serializable**
- **vs. H-Store, SILO, Shore-MT**
  - (Traditional DBs (e.g., MySQL) are too slow to compare with.)
- **HP Superdome X (DragonHawk):**
  - 16 sockets, 240 cores, 12TB DRAM.
- **Emulated NVRAM**
**Experiment 1: vs In-Memory DBMS**

**Observations**
1. SILO-style OCC much more resilient to contention
2. **100x~** faster than H-Store.
3. More Cores = More Speedup

<table>
<thead>
<tr>
<th>Cores</th>
<th>Speed-up [/H-Store]</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>33x</td>
</tr>
<tr>
<td>60</td>
<td>66x</td>
</tr>
<tr>
<td>240</td>
<td>394x</td>
</tr>
</tbody>
</table>

![Graph showing TPC-C Throughput vs Remote Transactions]
**Experiment 2: DBMS on NVRAM (emulated)**

**Environments**
- tmpfs-based NVRAM emulation (varied latency)
- Data/xlog in NVRAM
- H-Store w/ anti-cache

**Observations**
1. FOEDUS 100x~ faster
2. Performs best when NVRAM read <10us
**Experiment 3: OLAP Workload**

**Workload**
- 30x Larger Data
- Cursor-Heavy
- Read-Only
- Volatile Pages Only vs Snapshot Pages Only

**Observations**
1. 100x~ faster than H-Store.
2. FOEDUS runs faster when database is cold (!!)

*In Paper: how Dual Pages achieve it*
Ideas

• The key idea of FEODUS is to use the master-tree model to reduce contention, making OCC lightweight. Can this be extended to distributed memory systems with shared nothing topology, especially where reading remote memory is not very costly, like RDMA?
Ongoing Work

✓ Try on 1,000 Cores~
✓ Further Resilience to Contentions
  Combining Pessimistic Approach When Beneficial
✓ SIMD, Xeon-Phi
  Log-Gleaner’s Sorting, Page Construction, etc
✓ Non-C++ Interface
  SQL/JDBC/ODBC for OLAP. But, for OLTP???
Easy OCC with Foster Twins

Simple, Robust, and Efficient Search:

Find a pointer that *probably* contains the key;
Follow the page pointer;
*If* the page’s *key-range* contain the key: go on;
*Else*: *locally* retry;

No hand-over-hand verification/split counters.
No unnecessary local retry.
Absolutely no global retry.

• No Page-Version-set nor unnecessary aborts: All Records/TIDs always track-able via Foster-Twin!