Write-Behind Logging

Paper by Joy Arulraj, Matthew Perron and Andrew Pavlo

Made by Kjetil Kværnum
Based on presentation by Jørgen Aunet Halvorsen
HDDs

- Durable storage device
- Fast sequential access to data (block-oriented)
- High capacity
- Low price

- Slow for random access
SSDs

Pros
- Durable storage device
- Fast sequential access (block-oriented)
  Up to 1000x faster than HDDs
- Almost high capacity

Cons
- Slow for random access
- Fixed number of writes
- 3-10x more expensive per GB
DRAMs

- Fast access to fine-grained data (byte-level)
- Up to 1000x faster than SSDs
- Low capacity
NVMs

- Non-volatile memory
- Durable Storage device
- Fast access to fine grained data (byte-level)
- High capacity

More expensive per GB than HDDs
NVM vs SSD vs HDD

(a) Sequential Writes

(b) Random Writes
Using NVM in DBMS

How to use NVM in a DBMS running on a hybrid storage hierarchy with both DRAM and NVM?

Earlier approaches:

1. Optimizing the storage methods for NVM
   ○ Improves both the DBMS performance and the lifetime of the storage device
   ○ However, cannot be employed in a hybrid storage hierarchy, as they target an NVM-only system

2. Using NVM only for storing the log and managing the database still on disk
   ○ A more cost-effective solution
   ○ Only leverages the low-latency sequential writes of NVM
   ○ Does not exploit its ability to support random writes and fine-grained data access
Write-Ahead Logging (WAL)

- ARIES protocol, the most well-known recovery method based on WAL
- Our discussion is focused on disk-oriented DBMSs that use the multi-version concurrency control (MVCC)
- During normal operations, the DBMS records transactions’ modifications in a durable log before transferring data to database in disk
- To recover a database after a restart, the DBMS periodically takes checkpoints at runtime
Dirty Page Table (DPT)

- A disk-oriented DBMS maintains two metadata tables at runtime that it uses for recovery
- Dirty page table (DPT) & Active transaction table (ATT)
- DPT contains the modified pages that are in DRAM but have not been propagated to durable storage
- Each modified page has an entry in the DPT that marks the log record’s LSN of the oldest transaction that modified it
- To restore the page (Redo) during recovery
Active Transaction Table (ATT)

- The second table tracks the status of the running transactions.
- This table records the LSN of the latest log record of all active transactions.
- To undo the changes during recovery.
Disadvantages of WAL

- Although WAL supports efficient transaction processing when durable storage cannot support fast random writes, it is inefficient for NVM storage.
- The DBMS first records the tuple’s contents in the log, and it later propagates the change to the database.
- The logging algorithm can avoid this unnecessary data duplication.
Group Commit

- Most DBMSs use group commit to minimize the I/O overhead
- It batches the log records for a group of transactions in a buffer
- Then flushes them together with a single write to durable storage
- Improves the transactional throughput
Write-Behind Logging (WBL)

- The changes made by transactions are guaranteed to be already present on durable storage before they commit
- Dirty Tuple Table
- The recovery algorithm can scan the entire database to identify the dirty modifications

<table>
<thead>
<tr>
<th>Checksum</th>
<th>LSN</th>
<th>Log Record Type</th>
<th>Persisted Commit Timestamp ($C_p$)</th>
<th>Dirty Commit Timestamp ($C_d$)</th>
</tr>
</thead>
</table>
Write-Behind Logging (WBL)

- Tracking two commit timestamps in the log
  - It records the timestamp of the latest committed transaction whose changes and updates of prior transactions are safely persisted on durable storage ($c_p$).
  - It records the commit timestamp that the DBMS promises to not assign to any transaction before the next group commit finishes ($c_d$, where $c_p < c_d$).
- When the DBMS restarts after a failure, it considers all the transactions with commit timestamps earlier than $c_p$ as committed.
- Ignores the changes of the transactions whose commit timestamp is later than $c_p$ and earlier than $c_d$.
- In other words, if a tuple's begin timestamp falls within the ($c_p$, $c_d$) pair, then the DBMS's transaction manager ensures that it is not visible to any transaction that is executed after recovery.
Write-Behind Logging (WBL)
Write-Behind Logging (WBL)

- To reduce overhead, the DBMS’s garbage collector thread periodically scans the database
- Garbage collector undoes the dirty modifications associated with the currently present gaps
- Once all the modifications in a gap have been removed by the garbage collector, the DBMS stops checking for the gap in tuple visibility checks and no longer records it in the log
WBL Recovery Protocol

- Each WBL log record contains all the information needed for recovery:
  - The list of commit timestamp gaps
  - The commit timestamps of long running transactions that span across a group commit operation
- The DBMS only needs to retrieve this information during the analysis phase of the recovery process
## WAL vs WBL

<table>
<thead>
<tr>
<th>Runtime Operation</th>
<th>Commit Processing</th>
<th>Checkpointing</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Execute the operation.</td>
<td>• Collect log entries from log entry buffers.</td>
<td>• Construct checkpoint containing after-images of visible tuples.</td>
</tr>
<tr>
<td>• Write changes to table heap on DRAM.</td>
<td>• Sync the collected entries on durable storage.</td>
<td>• Write out transactionally consistent checkpoint to durable storage.</td>
</tr>
<tr>
<td>• Construct a log record based on operation (contains after-image of tuple).</td>
<td>• Mark all the transactions as committed.</td>
<td>• Truncate unnecessary log records.</td>
</tr>
<tr>
<td>• Append log record to log entry buffer.</td>
<td>• Inform workers about group commit.</td>
<td></td>
</tr>
<tr>
<td>WBL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Execute the operation.</td>
<td>• Determine dirty tuples using the DTT.</td>
<td>• Construct a checkpoint containing only the active commit identifier gaps (no after-images).</td>
</tr>
<tr>
<td>• Write changes to table heap on DRAM.</td>
<td>• Compute $c_p$ and $c_d$ for this group commit.</td>
<td>• Write out transactionally consistent checkpoint to durable storage.</td>
</tr>
<tr>
<td>• Add an entry to the DTT for that modification (does not contain after-image of tuple).</td>
<td>• Sync dirty blocks to durable storage.</td>
<td>• Truncate unnecessary log records.</td>
</tr>
<tr>
<td></td>
<td>• Sync a log entry containing $c_p$ and $c_d$.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Inform workers about group commit.</td>
<td></td>
</tr>
</tbody>
</table>
Features of the System

- Intel Lab’s hardware emulator that contains two Intel Xeon E5-4620 CPUs (2.6 GHz)
  - Each with 8 cores and a 20 MB L3 cache
- 256 GB of DRAM
  - Dedicates 128 GB of DRAM for the emulated NVM
- HDD: Seagate Barracuda (3 TB)
- SSD: Intel DC S3700 (400 GB)

Benchmarks:
- YCSB and TPC-C
Evaluation

Figure 2: WBL vs. WAL – The throughput, recovery time, and storage footprint of the DBMS for the YCSB benchmark with the write-ahead logging and write-behind logging protocols.
Figure 14: YCSB Throughput – The throughput of the DBMS for the YCSB benchmark with different logging protocols and durable storage devices.