ECS 165A Milestone 3: Multi-threaded, in-memory, and durable L-Store

“Nameless DB”
- Nicholas Chan
- Kevin Pack
- Anirudh Shenai (Ani)
- Jay Titterud
- Qing Zhou (Dave)
Database Architecture

Latched shared structures: **Index**, **Lock Manager**, **Bufferpool**
Revamped How NamelessDB Write to Disk

Previous Format
Page ranges stored as files

- Database
  - Table
    - Range 0
    - Range 1
    - Page directory
    - Metadata

M3 Format
Base and tail pages are stored as files

- Database
  - Table
    - Page Range
      - Base page data
      - Base page metadata
      - Tail page data
      - Tail page metadata
    - Page directory
    - Metadata
Simplify Page Directory

**Older implementation:**

Range index \[\rightarrow\] Range path

**Merge:** Originall had page range granularity. Calling a merge would update the range path to that of the merged range.

**New implementation:**

Range index \[\rightarrow\] Version array

The version array contains the merge version number of each base page

**Merge:** Now has base page granularity. Calling a merge increments the version number of the base page which is being merged.
Adjusted key structure to help query performance on columns featuring large amounts of duplicate keys

Queries Optimized:

\[ [1,20], [1,21] \rightarrow [1,[20,21]] \]

*Index.compress_key:*

Searches for pre-existing key value, appending the subsequent rid.

*Index.mini_delete:*

In the case where rid list contains more than one rid, the delete function just pops the appropriate rid from list, maintaining tree balance.
Update

Ensures accurate searching by updating existing BTrees after table data is altered. Takes place with write queries: Update, Insert, Delete.

**self.table.index.indices:**

[Index, None, None, Index, None]

<table>
<thead>
<tr>
<th>M2 Average</th>
<th>M3 Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update</td>
<td>Update</td>
</tr>
<tr>
<td>10.44387</td>
<td>6.703125</td>
</tr>
</tbody>
</table>
Merge will trigger when tail is page full but we only execute the merge on full base pages.

- Update the TPS column
- Update version control on Page Directory
- Preserves isolation and consistency
M3 Implementation for Concurrency

- Database
  - Page Directory
    - Page Range
      - Base/Tail Page
        - Page
  - Table
  - Index
  - Transaction Worker
  - Transaction
    - Lock Manager
    - Query
  - Bufferpool
Transaction Worker

**Exclusive**

- Sun

**Shared**

- Sun

Acquire a shared lock first for reading a record

If any transaction fails, the transaction gets aborted and rolled back

Only one exclusive lock for each record

Assign Transactions

Multiple shared lock for one record

Acquired a exclusive lock for write
Transaction

1. Attempts to execute

1. If any query fails, the transaction gets aborted and rollback

Transactions (List of Queries) → Generate → DBMS → Access

Users

Transaction Workers
Lock Manager

When the transaction worker acquires lock, the RID is managed by our Lock Manager.

Ensure no shared lock when there is an exclusive lock.

& Maintain a set of readers and up to one writer for each record (RID).
Strict 2PL

Transaction Workers

Acquire Locks

Release Locks

No wait 2PL

Exclusive

Shared
M3 Performance by Threading

- Window 10 OS
- Intel i7 @ 2.60GHz
- 16GB RAM

*1k queries function
*Used LRU eviction policy
M3 Performance by Eviction Policy

- Window 10 OS
- Intel i7 @ 2.60GHz
- 16GB RAM

*10K was used for all function except Aggregate
*Aggregate 100 of 100 record batch took
*Default 8 threads
M3 Performance by Eviction Policy

Average of 100 Run LRU

Query Function

Average of 100 Run MRU

Query Function
M3 Performance by Eviction Policy

Average of 100 Run MRU

Query Function

Insert | Update | Select | Aggregate | Delete
---|---|---|---|---
0 | 0 | 0 | 0 | 0

Time (sec)

Average of 100 Run MRU

Query Function

Insert | Update | Select | Aggregate | Delete
---|---|---|---|---
0 | 0 | 0 | 0 | 0

Time (sec)
Experiments in Each Milestone

- Big Page (Negligible impact on performance)
- Designed a rudimentary Index for key to RID pair
- Designed a rudimentary page directory that maps RIDS to records
- BufferPool Eviction Policies (LRU is most consistent in performance)
Conclusion

- Learned how to build a database from scratch, including implementation of queries, indexing, disk storage, merge, and multi-threading.

- Worked on testing including overall test, unit tests, and performance tests, as well as experimental design, debugging as we programmed.

- Developed a team work ethic which involved division of work, accountability and communication, and understanding different parts of the software design process.

- **We would like to thank Dr. Sadoghi and the TAs** for giving us this challenging assignment which pushed our creative limits and improved our skills. We are also grateful for all the help we received through the process.