L-Store Milestone 3

Alana Rufer, Eseosa Omorogieva, Nina Gopal,
Riddhi Barbhaiya, Kushaal Rao
Lock Manager
LockManager

Manages shared and exclusive locks used by 2PL
Coordinates release of all locks held by a transaction
Locks initialized during db.open() to reduce performance overhead of creating locks

![LockManager Diagram]

- **locks**
  - RID1: lock information
  - RID2: lock information
  - RID3: lock information
  - RID4: lock information
  - RID5: lock information
  - RID6: lock information
  - RIDN: lock information
  - ...

- **t_locked_rids**
  - TID1: [array of rids locked]
  - TID2: [array of rids locked]
  - TID3: [array of rids locked]
  - TID4: [array of rids locked]
  - TIDM: [array of rids locked]
  - ...

- **manager_lock**
- **t_lock**
Locking logic (for records)

If requested lock is not available, transaction immediately aborts (NoWait)

Lock() object provides thread-safety for lock information.

Actual holding of lock stored in type of lock and array of transaction TIDs

```
RID :

- threading.Lock()
- type of lock (S or X)
- [array of TIDs of transactions holding the lock]
```

Locks currently held on specified record

<table>
<thead>
<tr>
<th>Lock requested by Xact</th>
<th>None</th>
<th>S lock (same Xact)</th>
<th>X lock (same Xact)</th>
<th>S lock (different Xact)</th>
<th>X lock (different Xact)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S lock</td>
<td>acquire</td>
<td>no action</td>
<td>no action</td>
<td>acquire</td>
<td>abort</td>
</tr>
<tr>
<td>X lock</td>
<td>acquire</td>
<td>upgrade (if no other Xacts holding the lock) else abort</td>
<td>no action</td>
<td>abort</td>
<td>abort</td>
</tr>
</tbody>
</table>
Latch
Latch overview

Conflicts from accessing or modifying shared data

threading.Lock() for basic locking

Readers-Writer Lock: increase concurrency by allowing multiple readers when possible

Ex: two transactions insert at the same time (without latching):
Preventing Conflicts in Shared Data Structures

RID generation

page_directory: accessing/modifying entries

LogicalPage: accessing/modifying pageids

Index: accessing/modifying entries

Merge: starting merge, adding to merge list

db & transaction & create_index:

Readers-Writer Lock (writer-preferring)
“writers”: create_index, db.close()
“readers”: transactions
Guarantee success of create_index()
Wait for transactions to finish before db.close()

Insertion of record:
Prevent conflict on location in page, or double-creation of new page
Separate locks for base page and each active tail page

Bufferpool:
file pointer seek + r/w
accessing sensitive operations and data structures
S and X lock acquisition during queries

Select
- locate RID(s)
- (for each rid)
- acquire S lock on base record
- Read tail record?
  - yes
    - acquire S lock on tail record
  - no
    - assemble record

Insert
- construct record
- find location to insert
- acquire X lock on tail record
- tail record exists?
  - no
    - acquire X lock on new record
    - insert record
    - update index and page directory
  - yes
    - acquire X lock on base record
    - acquire X lock on tail record
    - insert record

Delete
- locate base record
- acquire X lock on base record
- invalidate base record
- acquire X lock on tail record
- invalidate tail record
- (for each rid)
- Update
  - locate RID
  - tail record exists?
    - yes
      - acquire X lock on tail record
      - create new tail record
    - no
      - acquire X lock on new record
      - Update index
      - Write update to records
Transaction & Transaction Worker
Transaction Worker

Transaction

<table>
<thead>
<tr>
<th>QUERY: Insert, Writes, Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY N</td>
</tr>
<tr>
<td>QUERY 4</td>
</tr>
<tr>
<td>QUERY 3</td>
</tr>
<tr>
<td>QUERY 2</td>
</tr>
<tr>
<td>QUERY 1</td>
</tr>
</tbody>
</table>

query_stack

- QN_info
- ...
- Q3_info
- Q2_info
- Q1_info

query_table

- Table N
- Table 3
- Table 2
- Table 1

Transaction Worker 1

thread 1

- Transaction 1
- Transaction 2
- Transaction N

Transaction Worker 2

thread 2

- Transaction 1
- Transaction 2
- Transaction N

Transaction Worker N

thread N

- Transaction 1
- Transaction 2
- Transaction N

time
Atomicity & Isolation

Serializability via **2PL**

If transaction is successful it is committed to the database

On failure to acquire a lock, transaction aborts immediately. All changes are rolled back

After transaction commits or aborts, all record locks held by the transaction are released (transaction atomicity)
Abort Transactions

Store info about completed transactions and their corresponding tables in two stacks

During the abort, we pop the stacks and undo each query

Release all rid locks during commit and abort

Query Info

Insert: (Success_value, "I" , inserted_rid, columns)
Delete: (Success_value, "D" , invalidated_rids, fields)
Update: (Success_value, "U" , only_locks, rid_to_update, last_update_rid, new_rid, old_schema_encoding, old_values, new_values, columns_modified)
**insert_undo**

Mark RID of inserted record as invalid

Remove the RID from the index on the columns if exists

**delete_undo**

Restore the invalid RID of the base record

Add the RID to the index on the columns if exists
**update_undo**  
**Update:** (Success_value, “U”, only_locks, rid_to_update, last_update_rid, new_rid, old_schema_encoding, old_values, new_values, columns_modified)
Performance
Commit Rates with Varying Contention

**Workload**

- 2 threads
- 25 transactions per thread
- update 1000 times (randomly choose the key to update)
- Vary the number of records in the db
  - Less records - more contention between transactions

Hardware: Dual-Core Intel Core i7, 2.5GHz, 16GB, 4 MB L3 Cache
Q&A