Efficient Lazy Timestamping in BerkeleyDB

Student: Shilong (Stanley) Yao
Advisor: Dr. Richard T. Snodgrass

Qualifying Oral Exam
Computer Science Department
University of Arizona
04/18/03 (1:30-3:00pm)

Outline
• Overview
• BerkeleyDB Architecture
• Stamper Design & Implementation
• Performance
• Summary

Efficient Lazy Timestamping in BerkeleyDB

Temporal Database
• Temporal database
  - Support some aspects of time
  - Simplify sophisticated queries over time
• Almost all database applications concern time
  - Academic: Course schedule over time
  - GIS: Land use over time
  - Accounting: Bill management over time
  - Etc.

A Challenge
• What’s Bob’s salary history?

CREATE TABLE Temp (Salary, Start, Stop)
AS SELECT salary, Start, Stop
FROM Employee
WHERE Name = 'Bob';

SELECT DISTINCT F.Salary, F.Start, F.Stop
FROM Temp AS F, Temp AS L
WHERE F.Start < L.Stop
AND F.Salary = L.Salary
AND NOT EXISTS (SELECT *
FROM Temp AS M
WHERE M.Salary = F.Salary
AND F.Start < M.Start AND M.Start < L.Stop
AND NOT EXISTS (SELECT *
FROM Temp AS T1
WHERE T1.Salary = F.Salary
AND T1.Start < M.Start AND M.Start <= T1.Stop
AND NOT EXISTS (SELECT *
FROM Temp AS T2
WHERE T2.Salary = F.Salary
AND (T2.Start < F.Start AND F.Start <= T2.Stop) OR
(T2.Start < L.Stop AND L.Stop < T2.Stop))

Transaction Time & Valid Time
• Transaction Time: When the fact is stored as current in the database.
• Valid Time: When the fact is true in the modeled reality.

Example of Transaction Database

(2002-05-20) Insert

Name | Salary | Start | Stop
--- | --- | --- | ---
Bob | 60,000 | 2002-05-20 | UC

(2003-01-01) Update

Name | Salary | Start | Stop
--- | --- | --- | ---
Bob | 70,000 | 2003-01-01 | UC

(2003-03-10) Delete

Name | Salary | Start | Stop
--- | --- | --- | ---
Bob | 60,000 | 2003-03-10 | UC
Efficient Lazy Timestamping in BerkeleyDB

Transaction Time Stamping
- **Definition:**
  - "Timestamping" is providing the transaction time to the tuple's internal time fields.
- **Key Problem:**
  - Ensure transaction consistent timestamps.
  - All timestamps of the same transaction are identical.
- **Several Key Issues:**
  - Which time to choose as the transaction time?
  - When to do the stamping?
  - What information is needed for the stamping?

Choose the Transaction Time
- **Begin time of the transaction**
  - **Advantage:**
    - Transaction time is available whenever updating a tuple
    - Double visiting is not needed
  - **Disadvantage:**
    - Requires concurrency control scheme based on timestamp ordering.
    - Loses the superiority of conventional locking.
- **Commit time of the transaction**
  - **Advantage:**
    - Supports 2PL
    - Lower abort rate
  - **Disadvantage:**
    - Need a stamper ID as the place holder for the timestamp
    - Revisiting the records with stamper ID is necessary

When To Stamp?
- **Eager Timestamping**
  - Efficient in-memory stamping
  - Heavy I/O when 'steal' policy is in effect
  - No double visiting or timestamp table needed
- **Pure Lazy Timestamping**
  - Need a list of updated unstamped pages

Laziness of the Stamping
- **Definition:** "Laziness" of the stamping is the latency between the transaction commit and stamping.

Lazy Time Stamping
- **How to stamp after commit?**
  - Keep a Transaction Time table (TT for short)
  - Double-visit the record: one before commit, the other after commit.

Outline
- **Overview**
- BerkeleyDB Architecture
- Stamper Design & Implementation
- Performance
- Summary
Efficient Lazy Timestamping in BerkeleyDB

**BerkeleyDB Overview**
- Open-source “embedded” database library developed by UC Berkeley and distributed by Sleepycat
- Current release version: 4.1.x
- Not a relational database

**BerkeleyDB Architecture**
- Original Major Subsystems
  - Access Methods Subsystem
  - Memory Pool Subsystem
  - Transaction Subsystem
  - Locking Subsystem
  - Logging Subsystem
- New Subsystem
  - Temporal Subsystem

**Role of the STP Module**
- Maintain the TT and TL table
- Do the stamping (Replacing the stamper ID with the actual transaction time of the transaction)
- Making TT survive system crashes

**STP Module**

**Outline**
- Overview
- BerkeleyDB Architecture
- Stamper Design & Implementation
- Performance
- Summary
Efficient Lazy Timestamping in BerkeleyDB

### Tuple Categories
- According to transaction time status:
  - type-1: Unstamped & uncommitted
  - type-2: Unstamped & committed
  - type-3: Stamped
- According to the storage status:
  - In-mem
  - On-disk

### Tuple Type Evolution

#### TT Table
- TT (Transaction Time Table)
  - log/recovery protected
- Fields
  - Transaction ID: (EnvD, Txnid)
  - Transaction Time
  - In-memory Un stamped Page List
  - On-disk Un stamped Page Count

#### TL Table
- TL (Transaction Location Table)
  - non-logged, non-recoverable, 2-D table

### Data Structures
- TL (Transaction Location Table)
  - non-logged, non-recoverable, 2-D table
How to Construct TL:

- Add a BT Entry
  - For each page read into the memory
  - For each page created in the memory

- Construct In-memory Unstamped Page List for Each BT Entry
  - At a transaction commit, get the list of WRITE-locked pages from LOCK subsystem
  - Add a node in the 2-D table for each of these pages

Garbage Collecting the TT Table:

- Garbage Collecting the TT Entry is Necessary
  - TT is in memory data structure
  - TT grows as new transactions begin
  - TT entry lookup is faster if TT is smaller

- How to Garbage Collect a TT Entry
  - In-mem unstamped page list becomes empty
  - On-disk unstamped page count becomes zero

- Problem
  - Tuples containing transaction IDs may move among pages

Handling Tuple Movement:

- When Does “Tuple Movement” happen?
  - BTree page split / merge
  - Copying duplicate keys off the page

- Solution: Adjust the On-disk Page Count
  - Pages fed with new records are passed to STP to register the unstamped tuples in TL

Algorithm Overview:

- Transaction Begin/Abort/Commit
- Handling Tuple Movement
- Stamping at pgread/pgwrite/fget
- Buffer Free
- Log/Recovery
  - Adding new TT entry
  - Modifying TT [1].od_pgcnt
  - Backup TT at checkpoint
  - Rebuild TT at recovery
- Renovation

Transaction Begin

stp_txn_begin (DB_TXN *tid)

Add an TT entry for this transaction

Init the TT entry:
- transaction time = INVALID_TIME
- in-mem page list = EMPTY
- on-disk page count = 0

Transaction Abort

stp_txn_abort (DB_TXN *tid)

Garbage collect the TT entry for this transaction
Efficient Lazy Timestamping in BerkeleyDB

Transaction Commit

stp_txn_commit(DB_TXN *tid)
- Fill the transaction time
- Get the list of pages WRITE-locked by this transaction and add them to the TL as the in-memory page list of this transaction

Handling Tuple Movement

stp_addbt(void *addrp)
- If there is no entry for this page in the TL table, add one for it.
- Scan the page and register in TL table all the transactions that updated this page

Stamping When Reading a Page

stp_pgread(void *addrp)
- For each rec. in page *addrp
  - Do the correspondent operation and update TL

Stamping When Writing a Page

stp_pgwrite(void *addrp)
- For each rec. in page *addrp
  - Do the correspondent operation and update TL

Buffer Free

stp_bhfree(void *addrp)
- Delete this page’s BT entry and the row of nodes in TL
- Garbage collect the TT entries whose in-memory page list become empty and od_pgcnt is zero

Logging

- Snapshot TT at checkpoint
  - Put all TT entries with positive od_pgcnt sequentially into the log
- Adding new TT entry
  - Whenever a new TT entry is created, log it.
- Modifying TT [i].od_pgcnt
  - Whenever TT [i].od_pgcnt is decreased, log it.
**Efficient Lazy Timestamping in BerkeleyDB**

**TT Table Recovery**

```
stp_recover()
```

Restore the TT Table snapshot in the latest checkpoint log entry

Scan from the latest checkpoint toward the end of the log, modify the TT table according to the TT modification log entries

---

**Logging/Recovery Example**

![Logging/Recovery Example](image)

**Renovation**

- What is Renovation?
  - The process of asynchronously reading the on-disk pages and stamping them so that TT entries are garbage collected.

- Why Renovation:
  - TT Table is in memory data structure
  - TT Table grows as new transactions begin
  - TT Entries need garbage collecting

- "Renovation" is similar as the "Vacuum Cleaner" in POSTGRES

---

**How to Renovate?**

- Db_renovate Utility
  - Working in parallel with user applications
  - Sequentially scan the database so that STP can stamp them

---

**Design Decisions**

- Higher Level vs. Lower Level
- Per Process vs. Per Environment
- In-memory Stamping Strategy: Stamping at Commit vs. Stamping after Commit
- On-disk Unstamped Page Tracking: In-memory vs. On-disk Data Structure
- Renovation At checkpointing vs. Concurrent Process
- How to Identify the In-memory Unstamped Pages

---

**Outline**

- Overview
- BerkeleyDB Architecture
- Stamper Design & Implementation
- Performance
- Summary
Efficient Lazy Timestamping in BerkeleyDB

**Outline**
- Overview
- BerkeleyDB Architecture
- Stamper Design & Implementation
- Performance
- Summary

**POSTGRES**

<table>
<thead>
<tr>
<th>Feature</th>
<th>T-BerkeleyDB</th>
<th>BerkeleyDB</th>
<th>Postgres</th>
</tr>
</thead>
<tbody>
<tr>
<td>2PL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>WAL</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Transaction Time</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time Support</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transaction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Management</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transaction Time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Commit Time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TT Table</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Force at commit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Steal</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Future Work**
- Minimize the log flush overhead
- Use more efficient data structure to store TT and TL, such as hash table, AVL tree, etc.

**Summary**
- Choose commit-time as timestamp
- Maintain TT table for timestamping
- Keep TT stable with the aid of original LOGGING/RECOVERY system
- Use auxiliary data structure TL to aid TT
- Minimum I/O overhead

**References**
- Richard T. Snodgrass, Michael H Bohlen, Christian S. Jensen and Adreas Steiner, Transitioning Temporal Support in TSQL2 to SQL3
- Richard T.Snodgrass, Temporal Database, Lecture of CS630 Spring 2002
- Betty Salzberg, Timestamping After Commit, IEEE 1994
References (cont.)

- Lawrence A. Rowe, Michael R. Stonebraker, The POSTGRES Data Model, 13th VLDB, Brighton 1987