Recovery with Aries

Anastassia Ailamaki
http://www.cs.cmu.edu/~natassa

Database Recovery

- Faults and fault tolerance
  Gray & Reuter ch. 2)
- Recovery from system crash
  - Overview
  - ARIES Recovery Algorithm
    - Log and logging
    - Recovery phase: Analysis, REDO, UNDO
      Ramakrishnan & Gehrke ch. 20
- Recovery from media failures
  Ramakrishnan & Gehrke ch. 20

Storage Classification

Buffers for Disk Pages
(Volatile Storage in Main memory)

Actual Database
(non-volatile storage)

System Log
(stable storage)
Failure Classifications

- Synchronous
  - Trappable by the operating system
  - No loss of data of any kind
  - Possible causes: program/logic errors: e.g. segmentation fault, division by zero

- Asynchronous
  - System crash
    - Assume loss of all data on volatile storage
    - Possible causes: power failures, operating system errors
  - Media failures (Disk crash)
    - Loss of data on online and volatile storage
    - Possible causes: Damages to storage media, Human errors (e.g. accidental erase)

Stable Storage Implementation

- Stable storage: theoretically impossible
  - (but can be simulated => data loss extremely unlikely)

- Mirrored disk: write => two separate disks
  - One disk may be remote

- RAID guarantee: a failure of a single disk, even during data transfer, will not result in data loss.

Fault Tolerance Metrics

- MTTF: Mean-time To Failure
- MTTR: Mean-time To Repair
- Availability: MTTF/(MTTF+MTTR)
Objectives of DBMS Recovery

- Guarantee atomicity and durability
- For good performance:
  - minimize time to restore database to a consistent state (MTTR) (e.g. minimize redo's)
  - minimize overhead imposed on normal transaction operations, given that system crash does not happen frequently
  - trade-off: recovery time vs. normal running time

i.e., Goals

- Back out effects of uncommitted transactions
- Recover results of committed transactions
- Get consistent snapshot of the DB (as a result of above)

Approach

- Some concurrency control mechanism such as locking with fancier tricks on “hot spots” such as indices.
- DO-UNDO-REDO paradigm for log records
- Write Ahead Log (WAL) protocol
- Two-Phase Commit protocol for distributed transactions
Overview of Database Recovery

- When a system crash occurs, some transactions could be in inconsistent state because they have not committed.
- A recovery system must be made aware of:
  - Transactions that were still running at the crash time
    - Those which have committed
    - Those which have not committed
  - Transactions that have committed will be allowed to finish (REDO)
  - Transactions that have not committed will be rolled back (UNDO)

Trade-offs

- Steal vs. no steal
- Atomic vs non-atomic
- Force vs. no force
- What’s the best solution?

Assumptions

- Concurrency control is in effect
- Updates are happening ‘in place’
- Steal, no force buffer management
- Read, Write as the database operations, which are atomic
- System crash may occur during database recovery
  - Apply redo and undo to a record either once only, or
  - Make redo and undo as idempotent operations
    - Idempotent: Acting as if used only once, even if used multiple times
System Model
(Normal Execution)

Transaction
Lock Request, Unlock
Scheduler
Buffer Manager
Read/Write
Get/release a page
Lock Request, Unlock

Scheduler
Buffer Manager
Read/Write
Get/release a page

Log
Database
Stable Storage

System Model
(System Restart after Crash)

Locking is usually not necessary.

Recovery Manager
Buffer Manager
Read/Write
Get/release a page

Buffer Manager

Log
Database
Stable Storage

ARIES

- ARIES is used the dominant crash recovery algorithm in commercial DBMS
- Three principles:
  - Write-ahead logging
  - Repeating history during REDO
  - Logging changes during UNDO
Recovery Passes over the Log

- Analysis pass: collects info about dirty pages and uncommitted transactions. It starts at the most recent checkpoint, going forward.
- REDO pass: “repeats history”, redoing updates from the earliest spot in the log where an update might have been lost.
- UNDO pass: goes backward from end of log, removing the effects of all uncommitted updates from the DB.

ARIES Recovery Algorithm

```
<table>
<thead>
<tr>
<th>“start” of oldest in-progress transaction</th>
<th>First possibly lost update to do crash</th>
<th>Most recent (known) checkpoint</th>
<th>End of Log</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1) Analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2) REDO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3) UNDO</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```