Welcome to ACS TA session 4

Alexander, Lasse, Svend, Yiwen, and Yuan

Department of computer science, University of Copenhagen

Academic year 2018-2019, Block 2

Agenda for today

- **Feedback on Assignment 2**
- **Revisit ARIES**
  - ARIES properties, approach, principles, data structures, phases
  - Exercises on the ARIES algorithm
- **Reliability questions**

Theory Assignment 1: Feedback

Question 1.1:

Concurrency ≠ Parallelism

- Parallelism: dividing one task into multiple subtasks, which are executed in parallel.
- Concurrency: multiple tasks running in the same system before, after or during each other.

Question 2.1:

Centralized components which map the memory space are bad, what happens if it fails and it can be highly congested.

Question 2.3:

Atomicity for single memory operations ≠ Atomicity for transactions.

Question 3.2:

Asking whether S2PL could cause it, not to rewrite them such that S2PL did cause them.

Programming Assignment 2: Single Lock

removeAllBooks: many groups missed this method,
remember to implement all the methods

validate (!):

the validate phase needs to be protected (why?)

getEditorPicks:
many groups released the lock before they completed the gathering phase,
which can lead to dirty reads or returning books that are already removed from the book store

Use this pattern

```
lock here
 try {
   return value (if needed)
 }
 finally {
   unlock here
 }
```

instead of

```
lock here
 if( should throw exception 1) {
   unlock
   throw
 }
 if( should throw exception 2) {
   unlock
   throw
 }
 unlock
 return
```

Programming Assignment 2: Two Level Lock

- No more locks are required if an exclusive lock is granted on the top.
- Use Try-Finally clause to release locks instead of Try-Catch.
- Strict two-phase locking is vulnerable to deadlocks
- Leaf-to-root lock release

Programming Assignment 2: Tests

- **Test1:** operations on single book verifies **only before-or-after not all-or-nothing** atomicity. It is okay but a poor test case, thus at least 2 books are suggested.
- **Test2:** single book fails to verify snapshot consistency, using at least 2 books.
- **Deadlock:** many submissions try to verify deadlocks would not happen, but it is very difficult. When deadlocks happen with high probability, it is necessary to detect deadlocks and fail the test.

Revisit ARIES: principles and properties

Properties

- **Atomicity**: undo the transactions that do not commit
- **Durability**: ensure all actions of committed transactions survive system crashes and media failures

Approach

- **Steal**: pages are written to disk in yet uncommitted transactions
- **No-force**: when a transaction commits, its pages are not forced to disk

Principles

- **Write-ahead logging**: log the operation before executing it
- **Repeat history**: re-bring system to its state when it crashed and then fix
- **Log the undo**: to fully repeat the history, including the undo operations.

BUT we never undo the undo operations.

ARIES log record data structure

Log record

- **Log**: chronologic sequence of log entries
- **Log tail**: the portion of the log in main memory (not forced yet)
- **Log sequence numbers** (LSN): strictly increasing IDs for log records

Log record types and fields

- ***All**: prevLSN, transID, type*
- **Update**: pageID, length, offset, before-image, after-image
- **Commit**, **Abort**, **End**
- **Compensation** (CLR): undoNextLSN

ARIES additional data structures

Dirty pages table

- One entry per page not written to disk yet
- **Fields**: pageID, recLSN

Transactions table

- One entry per transaction
- **Fields**: transID, lastLSN, status
- Entries with status **committed** or **aborted** are removed from the

Table when the corresponding transaction reaches the **end** state

ARIES phases

Analysis

- Identifies dirty pages and active transactions

Redo

- Repeats all actions from safe point to moment of crash
- Leaves the data structures in the latest state prior to the crash

Undo

- Undoes the actions of uncommitted transactions reverse-chronologically

ARIES Questions

After a crash failure, where in the log ...

1. should the **analysis** phase start? begin_checkpoint, checkpoints not necessary atomic
2. should the **redo** phase start?
3. should the **undo** phase end?

Exercise 1

Apply the ARIES recovery algorithm to the next scenario. Show:

1. the state of the transaction and dirty page tables after the **analysis** phase
2. the sets of winner and loser transactions
3. the values for the LSNs where **redo** starts and **undo** ends

How far back into the log must ARIES scan during **redo** and **undo**?

What are:

1. the set of log records that may cause pages to be rewritten during redo?
2. the set of log records undone during undo?
3. the contents of the log after the recovery procedure completes?

Exercise 1

TransID	Status	LastLSN
T2	active	2
T1	committed	3

Transactions

LSN	PrevLSN	TransID	Type	PageID
1	0	T1	update	C
2	0	T2	update	B
3	1	T1	commit	
4 5			begin checkpoint end checkpoint	
6	3	T1	end	
7	0	T3	update	A
8	2	T2	update	C
9	8	T2	commit	
10	9	T2	end	

Crash

PageID	RecLSN
C	1
B	2

Dirty pages

Exercise 2

The next scenario depicts a situation where the system crashes during recovery. Apply the ARIES algorithm after:

1. **Crash 1** that occurred during normal execution
2. **Crash 2** that occurred during recovery

Exercise 2

LSN	PrevLSN	TransID	Type	undoNextLSN	PageID
01 05			begin checkpoint end checkpoint		
10	-	T1	update		P5
20	-	T2	update		P3
30	10	T1	abort		
40 45	30 40	T1	CLR: Undo LSN 10 end	-	
50	-	T3	update		P1
60	20	T2	update		P5
Crash 1					
70	60	T2	Abort		
80	50	T3	Abort		
90	70	T2	CLR: Undo LSN 60	20	
100 105	80 100	T3	CLR: Undo LSN 50 end	-	
Crash 2					
110 115	90 110	T2	compensate LSN 20 end	-	

Faults, Errors, Failures. Fault tolerance

Faults, errors, failures

- Fault: defect that has the potential to cause problems
- Error: wrong result caused by an active fault
- Failure: unhandled error that causes the interface to break its contract

Fault tolerance

- Error detection: verify correctness with some redundancy, e.g., **fail fast**
- Error containment: limit the propagation of errors, e.g., **fail stop/safe/soft**
- Error masking: ensure the correct operation despite errors

Reliability Questions - true or false?

1. A **fault** is an unhandled error that causes an interface to break its contract.
2. A **fail-fast** component is one that immediately stops when an error occurs in order to prevent error propagation.

Thank you

Alexander, Lasse, Svend, Yiwen, and Yuan
Department of computer science, University of Copenhagen

Academic year 2018-2019, Block 2