Introduction to Project 3

Discussion 7

Project 3 Goals

- Understanding the ARIES recovery algorithm
- Implementing the ARIES recovery algorithm in C++

Getting Started

- Read the recovery sections of the textbook:
  - Section 16.7 – Introduces the recovery manager
  - Chapter 18 – Covers the ARIES recovery Algorithm
    - Super important to read and understand this section because this is what the project wants you to implement

Getting Started

- Download the zip file contained in the project description on Canvas
- Understand the different components of the recovery simulator
  - LogRecord
  - Storage Engine
  - Main
  - LogMgr
    - This is what you will implement for the project

Grading

- You can access the autograder at https://grader484.eecs.umich.edu/
- Autograder beta version is now open
- Limit of 4 submission per day
- Autograder runs hidden test suite using your LogMgr.cpp and checks the output
- Highest score will be the final grade
  - If you submit late, the penalty is only applied to the late submission

The Disk

- The disk is represented as a simple text file
- Each line corresponds to a page
- Each page consists of an LSN followed by a string which is the data stored on the page
- An example of the disk is located in StorageEngine/sampleDBFile.txt

Testcases

- testcases/ contains 6 basic test cases to help check that your project is working
- Passing all 6 tests cases does not mean you will pass the autograder. Create your own test cases as well.
- Results of testcases can be found in output/log and output/dbs/
    - Remember to remove the files if you run the case again

Testcase Format

- First line of the file tells the simulator where to find the text file representing the initial state of the disk
- All lines after specify an action to perform
    - Write - txid, action, pageid, offset, data
    - Commit
    - Abort – txid, action, # of writes allowed during abort
    - Checkpoint – action
    - Crash – action {# of writes allowed during crash}
    - end marks the end of the transaction

Running Project 3

- In the parent direction, run make
  - You may want to modify the Makefile to do more complex operations such as running all the testscases
- From the parent direction, run the simulator with the command: ./main.o testcases/test00
  - This will run test00

LogRecord.h

- Contains txTableEntry struct
  - Create new entries by calling txTableEntry(lsn, status) constructor
- Contains four log record classes:
  - LogRecord
    - Base log record type
  - UpdateLogRecord
    - Log record for updates
  - CompensationLogRecord
    - Log Record for CLR
  - ChkptLogRecord
    - Log Record for checkpoint operations

LogRecord.cpp

- Two functions we care about here:
  - toString() – This will convert a log record of any type into a string.
  - stringToRecordPtr – This will convert a string corresponding to a single log record into a LogRecord pointer

Storage Engine

- This keeps track of pages and manages disk access
  - Adds pages to in-memory buffer if read/write performed
  - Flushes pages to memory if buffer is full
- A page consists of an id, an LSN, a dirty bit, and a string of data.

Storage Engine

- Functions that may be useful:
  - nextLSN() - This will return the a unique ID assigned in monotonically increasing order
  - pageWrite(…) – This will write a page into memory
    - This can return false if you are no longer allowed to write pages. This is determined by page_write_permitted.
    - Normally the storage engine will do this, but this is needed for aborts or recovery

Storage Engine

- Functions that may be useful:
  - getLSN(…) - This will return the lastLSN of a the given page id
  - store_master(…) – This writes an int to a location in disk
  - get_master() – This retrieves the int written by store_master
  - getLog() – Gets the log from disk

Main

- This reads in the testcase files and initializes the log manager and storage engine
- It also parses the testcase file and calls the appropriate functions

LogMgr

- This is the file you are turning in for the project
- Implement all the functions specified in LogMgr.h
- You must maintain the transaction table and the dirty page table. These are stored as maps within LogMgr.h

Debugging the Code

- Using gdb on the CAEN machine will probably have issues displaying certain data structures like maps.
- One possible solution :
  - mkdir <directory_name>
  - cd <directory_name>
  - svn co svn://gcc.gnu.org/svn/gcc/branches/gcc-4_6-branch/libstdc++-v3/python
  - vim ~/.gdbinit
  - Add the code on the next slide to gdbinit

Code to add to gdbinit

- Replace /path/to/<directory_name>/python with the actual filepath

-
  ```
  python
  import sys
  sys.path.insert(0, '/path/to/gdb_printers/python')
  from libstdcxx.v6.printers import register_libstdcxx_printers
  register_libstdcxx_printers (None)
  end
  ```

Things to note

- **DO NOT** modify any file other than LogMgr.cpp
- **DO NOT** directly read/write to disk. You must use the StoreEngine interface to do this
- **DO NOT** create static variables or functions in LogMrg.cpp
- **DO** get started early. Get something ready for when the autograder opens
- **DO** read the textbook chapters. Chapter 18 contains most of the information you need to do the project

Things to note

- In the stringToLRVector function, add:

```
vector<LogRecord*> result;

istringstream stream(logstring);
string line;
while(getline(stream, line)) {
        LogRecord* lr = LogRecord::stringToRecordPtr(line);
        Results.push_back(lr);
}
return result;
```

Questions about the Project?

Example

- 1 write 2 0 one
- 2 write 3 0 two
- 2 commit
- 3 write 1 0 three
- 3 write 2 0 four
- crash {7}

Stable Storage

Disk State

-1 xxxxxxxxxx

-1 xxxxxxxxxx

-1 xxxxxxxxxx

-1 xxxxxxxxxx

Log State

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After

T1 writes

- 1 write 2 0 one

Transaction Table

transID	lastLSN	Stat
1	2	U

Log tail

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After
2	-1	1	update	2	0	xxx	one

Dirty Page Table

pageID	recLSN
2	2

T2 writes

- 2 write 3 0 two

Transaction Table

transID	lastLSN	Stat
1	2	U
2	3	U

Log tail

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After
2	-1	1	update	2	0	xxx	one
3	-1	2	update	3	0	xxx	two

Dirty Page Table

pageID	recLSN
2	2
3	3

T2 commits

- 2 commit

Transaction Table

transID	lastLSN	Stat
1	2	U
2	3	U

Log tail

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After
2	-1	1	update	2	0	xxx	one
3	-1	2	update	3	0	xxx	two
4	3	2	comm				

Dirty Page Table

pageID	recLSN
2	2
3	3

Stable Storage – After commit

Disk State

-1 xxxxxxxxxx

-1 xxxxxxxxxx

-1 xxxxxxxxxx

-1 xxxxxxxxxx

Log State

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After
2	-1	1	update	2	0	xxx	one
3	-1	2	update	3	0	xxx	two
4	3	2	comm				

After Commit

- 2 commit

Transaction Table

transID	lastLSN	Stat
1	2	U

Log tail

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After
5	4	2	end				

Dirty Page Table

pageID	recLSN
2	2
3	3

T3 Writes

- 3 write 1 0 three

Transaction Table

transID	lastLSN	Stat
1	2	U
3	6	U

Log tail

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After
5	4	2	end				
6	-1	3	update	1	0	xxxxx	three

Dirty Page Table

pageID	recLSN
2	2
3	3
1	6

T3 Writes

- 3 write 2 0 four

Transaction Table

transID	lastLSN	Stat
1	2	U
3	7	U

Log tail

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After
5	4	2	end				
6	-1	3	update	1	0	xxxxx	three
7	6	3	update	2	0	onex	four

Dirty Page Table

pageID	recLSN
2	2
3	3
1	6

Crash

- crash {7}

Transaction Table

transID	lastLSN	Stat

Log tail

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After

Dirty Page Table

pageID	recLSN

Analysis Steps

- Find the most recent begin checkpoint. If there isn’t one, then start from the beginning of the log
- Look at each record:
  - End log: Remove T from the transaction table
  - Others: Add T to the transaction table if not there.

    Change lastLSN

    If it is a commit, set status to C.

    If it is a redoable record on P, add P to the dirty page table

Analysis

- crash {7}

Transaction Table

transID	lastLSN	Stat

Log tail

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After

Dirty Page Table

pageID	recLSN

Disk Log

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After
2	-1	1	update	2	0	xxx	one
3	-1	2	update	3	0	xxx	two
4	3	2	comm				

Analysis

- crash {7}

Transaction Table

transID	lastLSN	Stat
1	2	U

Log tail

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After

Dirty Page Table

pageID	recLSN
2	2

Disk Log

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After
2	-1	1	update	2	0	xxx	one
3	-1	2	update	3	0	xxx	two
4	3	2	comm				

Analysis

- crash {7}

Transaction Table

transID	lastLSN	Stat
1	2	U
2	3	U

Log tail

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After

Dirty Page Table

pageID	recLSN
2	2
3	3

Disk Log

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After
2	-1	1	update	2	0	xxx	one
3	-1	2	update	3	0	xxx	two
4	3	2	comm				

Analysis

- crash {7}

Transaction Table

transID	lastLSN	Stat
1	2	U
2	4	C

Log tail

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After

Dirty Page Table

pageID	recLSN
2	2
3	3

Disk Log

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After
2	-1	1	update	2	0	xxx	one
3	-1	2	update	3	0	xxx	two
4	3	2	comm				

Redo Steps

- Find oldest update in log (smallest recLSN) and start at that point in the log
- For each redoable record:
  - Is the page in dirty page table?
  - The page is in dirty page table, but is recLSN for the page is less than or equal to the LSN of the record?
  - Is the pageLSN less than the LSN of the log record?
  - If it yes for all three, redo the record
- Remove committed transactions from table

Redo

- crash {7}

Transaction Table

transID	lastLSN	Stat
1	2	U
2	4	C

Log tail

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After

Dirty Page Table

pageID	recLSN
2	2
3	3

Disk Log

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After
2	-1	1	update	2	0	xxx	one
3	-1	2	update	3	0	xxx	two
4	3	2	comm				

Redo

- crash {7}

Transaction Table

transID	lastLSN	Stat
1	2	U
2	4	C

Log tail

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After

Dirty Page Table

pageID	recLSN
2	2
3	3

Disk Log

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After
2	-1	1	update	2	0	xxx	one
3	-1	2	update	3	0	xxx	two
4	3	2	comm				

Redo

- crash {7}

Transaction Table

transID	lastLSN	Stat
1	2	U

Log tail

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After
8	4	2	end				

Dirty Page Table

pageID	recLSN
2	2
3	3

Disk Log

LSN	prvLSN	transID	type	pageID	offset	before	After
2	-1	1	update	2	0	xxx	one
3	-1	2	update	3	0	xxx	two
4	3	2	comm				

Stable Storage – After redo

Disk State

-1 xxxxxxxxxx

2 xxxxxxxxxx

3 xxxxxxxxxx

-1 xxxxxxxxxx

Log State

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After
2	-1	1	update	2	0	xxx	one
3	-1	2	update	3	0	xxx	two
4	3	2	comm				

Undo Steps

- Undo all in the transaction table starting with the transaction with the largest LSN value in transaction table
- For each record:
  - If CLR:
    - If undoNextLSN is null, add end record to log
    - Otherwise, add undoNextLSN to the set to undo
  - If update:
    - Create a CLR record in the log. Add end record if undoNext is null
    - Add prevLSN to set to undo

Undo

- crash {7}

Transaction Table

transID	lastLSN	Stat

Log tail

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	after
8	4	2	end				
9	2	1	CLR	null			
10	9	1	end				

Dirty Page Table

pageID	recLSN

Disk Log

LSN	prvLSN	transID	type	pageID	offset	before	After
2	-1	1	update	2	0	xxx	one
3	-1	2	update	3	0	xxx	two
4	3	2	comm				

Stable Storage – After undo

Disk State

-1 xxxxxxxxxx

2 xxxxxxxxxx

3 xxxxxxxxxx

-1 xxxxxxxxxx

Log State

LSN	prvLSN	transID	type	pageID or undoNext	offset	before	After
2	-1	1	update	2	0	xxx	one
3	-1	2	update	3	0	xxx	two
4	3	2	comm				