E-66 Database Systems

Brendan Murphy

Problem Set 5

Problem 1: Logging and Recovery

1. In undo-redo logging values can go to disk at any time. If a crash occurs and log record 80 is the last one to make it to disk, **all** of the following are possible on-disk values for each data item after the crash but before the recovery:

A. 100, 130, 150
B. 500, 510, 570
C. 400, 430
D. 200, 210

2. In redo-only logging none of the changes make it to disk until the commit. Given the same scenario as above, but in the context of a system using **redo-only** logging, the possible on-disk values below reflect the fact that in **redo-only** logging database pages are held in memory until commit. The result is transaction 2`s values being lost because it never commits:

A. 100, 130, 150
B. 500, 510
C. 400
D. 200

3. Given the same scenario, but in the context of a system using **undo-only** logging, the possible on-disk values below reflect the fact that in **undo-only** logging when a transaction commits it forces even the dirty database pages to disk. The result is transaction 2`s uncommitted values possibly being on disk:

A. 150
B. 510, 570
C. 400, 430
D. 200, 210

4. If a crash occurs and log record 80 is the last one to make it to disk, the following steps would be performed in the context of a system that is performing **undo-redo** logging and the on-disk datum LSNs are not consulted.

Backward Pass:
LSN
80. undo: set item B = 510
70. undo: set item D = 200
60. add transaction 1 to commit list
50. skip
40. skip
30. undo: set item C = 400
20. skip
10. skip
0. skip

Forward Pass:
LSN
0. skip
10. redo: set item A = 130
20. skip
30. skip
40. redo: set item A = 150
50. redo: set item B = 510
60. skip
70. skip
80. skip

5. If a crash occurs and log record 80 is the last one to make it to disk, the following steps would be performed in the context of a system that is performing **undo-redo** logging and the on-disk datum LSNs *are* consulted.

Datum LSNs: A: 0, B: 50, C: 0, D: 70

Backward Pass:
LSN
80. B`s datum LSN = 50 != LSN of log record 80, no action taken
70. D`s datum LSN = 70 == LSN of log record 70, must undo. Set D=200 LSN = 0
60. add transaction 1 to commit list
50. skip
40. skip
30. C `s datum LSN = 0 != LSN of log record 30, no action taken
20. skip
10. skip
0. skip

Forward Pass:
LSN
0. skip
10. A`s datum LSN = 0 == 0 olsn of log record, must redo. Set A=130 LSN = 10
20. skip
30. skip
40. A`s datum LSN = 10 == 10 olsn of log record, must redo. Set A=150 LSN = 40
50. B`s datum LSN = 50 != 0 olsn of log record, no action taken
60. skip
70. skip
80. skip

6. If a dynamic checkpoint had occurred between log record 30 and 40, this would change my answer to part 4. There are two active transactions at the time of the checkpoint, transaction 1 and transaction 2. On the backward pass LSN 30 would now be a candidate for undo and C is set back to 400. On the forward pass redo LSN 40 setting A to 150 and LSN 50 setting B to 110.

Problem 2: Data Models for a NoSQL Document Database

1. Data modeling using references

song document

{

_id:'0123456780',

singer: [{singer_id: '9876543'}],

name: 'Brave',

duration: 219,

genre: 'pop',

best_chart_rank: 23,

royalties_due: 1200.00

}

artist document

{

_id:'987543',

name: 'Sara Bareilles',

label: 'Epic',

dob: '1979-12-07',

primary_genre: 'pop'

}

sings document

{

_id:'0123456780',

artist_id: '9876543'

}

played document

{

_id:'0123456780',

date:'2015-04-22',

time: '17.00'

}

2. One advantage of a reference approach to data modeling in part one is that it allows you to capture things like trees, hierarchies, and graphs. One disadvantage of this approach to data modeling is that system needs to make more requests to perform a given logical operation which can hinder performance. For example, for the system to retrieve the song and artist will require two queries versus one if the data were embedded.

3. Data Modeling Using Embedding:

song document

{

_id:'0123456780',

singer: [{singer_id: '9876543'}],

name: 'Brave',

duration: 219,

genre: 'pop',

best_chart_rank: 23,

royalties_due: 1200.00,

artist: { _id:'987543',

name: 'Sara Bareilles',

label: 'Epic',

dob: '1979-12-07',

primary_genre: 'pop'}

played:{ _id:'0123456780',

date:'2015-04-22',

time: '17.00'}

}

4. One advantage of an embedded approach to data modeling is that the system needs to make fewer requests for a given logical operation, this improves performance. One disadvantage of this approach to data modeling is the duplication of data and the possibility on inconsistencies between different versions of duplicated data. For example, if an artist were to change to a new label, this would require every song document to be updated with the new label name and there is an inherit risk of missing a song.

5. If using a reference data model, when a customer plays a song the played document gets updated with a new entry including the id of the song and the date and time it was played.

If using an embedded data model, when a customer plays a song a new played embedded document is created with the song id and the date and time from when it was played.

Problem 3: Two Phase Commit

1. If a site is using **undo-only** logging to commit a sub transaction, the following steps are required to put itself into a ready or precommitted state:

1. Receive prepare message from coordinator
2. Force all dirty log records to disk
3. Force any database pages changed by transaction to disk
4. Write the commit log record if needed
5. Force the commit log record to disk
6. Force write ready log to disk
7. Report ready state and await instruction from the coordinator

2. In a distributed DBMS that employs two-phase commit, the system is configured to wait for a failed coordinator to recover. This is how the coordinator would use its log records for a given distributed transaction T to decide what actions to take for that transaction during recovery.

1. Check for a commit record for T, if one exists commit T else go to step 4
2. Check for commit messages sent to other sites, if not send out those messages
3. Check sites for abort records and if they exist resend abort message
4. If there is no commit record for T, abort the transaction and force write an abort record
5. Send abort messages to the sub transaction sites
6. If all sub transaction sites report a ready state abort transaction T
7. Send abort messages to the sub transaction sites