Lecture 28:

Monday, December 9, 2002

1

Outline

- From the homework: Mr. Frumble's blues
- An exercise: counting the number of joins

- Redo logging – 17.3
- Redo/undo logging – 17.4

- Course evaluation forms

2

Understanding Hash Function Distribution

- N = 100 buckets
- Find the distribution of:
  H('a00'), H('a01'), …, H('a99')
- Ascii('a') = 97, ascii('0') = 48
- Hence all values will start with:
  (97+48+48) mod 100 = 93
  think of 93 as the new origin, and ignore it

3

Understanding Hash Function Distribution

- Hence the values of:
  H('a00'), H('a01'), …, H('a99')
  are:
  0+0, 0+1, 0+2, ... , 9+9

- Observation 1: only buckets 0, 1, …, 18 contain data !
- Observation 2:
  - Buckets 0 and 18 contain 1 data item
  - Buckets 1 and 17 contain 2 data items
  - . . .
  - Bucket 9 contains 10 data items
- Then what happens with H('a00000'), …, H('a99999') ?

4

Counting the Number of Join Orders (Exercise)

$R_0(A_0,A_1) \bowtie R_1(A_1,A_2) \bowtie \ldots \bowtie R_n(A_n,A_{n+1})$

- The number of left linear join trees is:
  $n!$
- The number of left linear join trees without cartesian products is:
  2^n (why ?)
- The number of bushy join trees is:
  $n!/(n+1)*C^{2n}_n = (2n)!/((n+1)*(n!))$
- The number of bushy join trees without cartesian product is:
  $2^{n-1}/(n+1)*C^{2n}_n$ (why ?)

5

Number of Subplans Inspected by Dynamic Programming

$R_0(A_0,A_1) \bowtie R_1(A_1,A_2) \bowtie \ldots \bowtie R_n(A_n,A_{n+1})$

- The number of left linear subplans inspected is:
  $\Sigma_{k=1,n}C^n_k*k = n2^{n-1}$

- The number of left linear subplans without cartesian products inspected is:
  $\Sigma_{k=1,n}(n-k+1)*2 = n(n+1)$ why ?

- The number of bushy join subplans inspected is:
  $\Sigma_{k=1,n}C^n_k*2^k = 3^k$ why ?

- The number of bushy join subplans without cartesian product:
  $\Sigma_{k=1,n}(n-k+1)*(k-1) = n*n*(n-1)/2 – n(n-1)(2n-1)/6 = n(n-1)(n+1)/6$

6

Redo Logging

Log records

- <START T> = transaction T has begun
- <COMMIT T> = T has committed
- <ABORT T>= T has aborted
- <T,X,v>= T has updated element X, and its *new* value is v

Redo-Logging Rules

R1: If T modifies X, then both <T,X,v> and <COMMIT T> must be written to disk before X is written to disk

- Hence: OUTPUTs are done *late*

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
REAT(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,16>
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,16>
						<COMMIT T>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	

Recovery with Redo Log

After system's crash, run recovery manager

- Step 1. Decide for each transaction T whether it is completed or not
  - <START T>….<COMMIT T>…. = yes
  - <START T>….<ABORT T>……. = yes
  - <START T>……………………… = no
- Step 2. Read log from the beginning, redo all updates of *committed* transactions

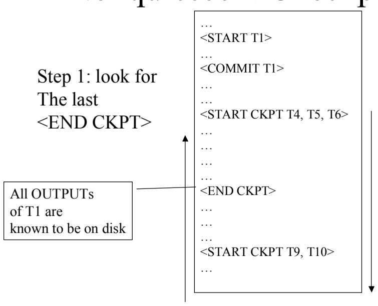
Recovery with Redo Log

<START T1>
<T1,X1,v1>
<START T2>
<T2, X2, v2>
<START T3>
<T1,X3,v3>
<COMMIT T2>
<T3,X4,v4>
<T1,X5,v5>
…
…

Nonquiescent Checkpointing

- Write a <START CKPT(T1,…,Tk)> where T1,…,Tk are all active transactions
- Flush to disk all blocks of committed transactions (*dirty blocks*), while continuing normal operation
- When all blocks have been written, write <END CKPT>

Redo Recovery with Nonquiescent Checkpointing



13

Comparison Undo/Redo

- Undo logging:
  - OUTPUT must be done early
  - If <COMMIT T> is seen, T definitely has written all its data to disk (hence, don't need to redo) – inefficient
- Redo logging
  - OUTPUT must be done late
  - If <COMMIT T> is not seen, T definitely has not written any of its data to disk (hence there is not dirty data on disk, no need to undo) – inflexible
- Would like more flexibility on when to OUTPUT: undo/redo logging (next)

14

Undo/Redo Logging

Log records, only one change

- <T,X,u,v>= T has updated element X, its *old* value was u, and its *new* value is v

15

Undo/Redo-Logging Rule

UR1: If T modifies X, then <T,X,u,v> must be written to disk before X is written to disk

Note: we are free to OUTPUT early or late (i.e. before or after <COMMIT T>)

16

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
REAT(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8,16>
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8,16>
OUTPUT(A)	16	16	16	16	8	
						<COMMIT T>
OUTPUT(B)	16	16	16	16	16	

17

Recovery with Undo/Redo Log

After system's crash, run recovery manager

- Redo all committed transaction, top-down
- Undo all uncommitted transactions, bottom-up

18

Recovery with Redo Log

<START T1>
<T1,X1,v1>
<START T2>
<T2, X2, v2>
<START T3>
<T1,X3,v3>
<COMMIT T2>
<T3,X4,v4>
<T1,X5,v5>
…
…