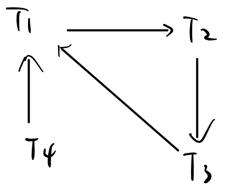
CS150A Homework 3 – Writing

School of Information Science and Technology
December 3, 2022

1 Concurrency Control(20 pts)

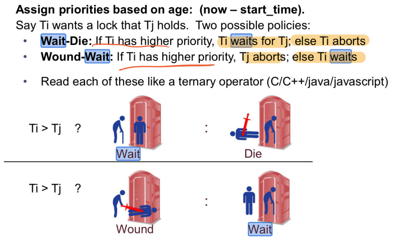
T1	S(A)	S(D)		S(B)					
T2			X(B)				X(C)		
T3					S(D)	S(C)			X(A)
T4								X(B)	

1. Draw the "waits-for" graph and explain whether or not there is a deadlock.



2. If we try to avoid deadlock by using the wait-die deadlock avoidance policy, would any transactions be aborted? Assume T1 < T2 < T3 < T4 for the priority.

T2 X(B) T1 S(B): wait, since T1 < T2
T3 S(C) T2 X(C): wait, since T2 < T3
T4 X(B) T1 S(B): abort T1, since T1 < T4.
T3 X(A) T1 S(A): abort T1, since T1 < T3.

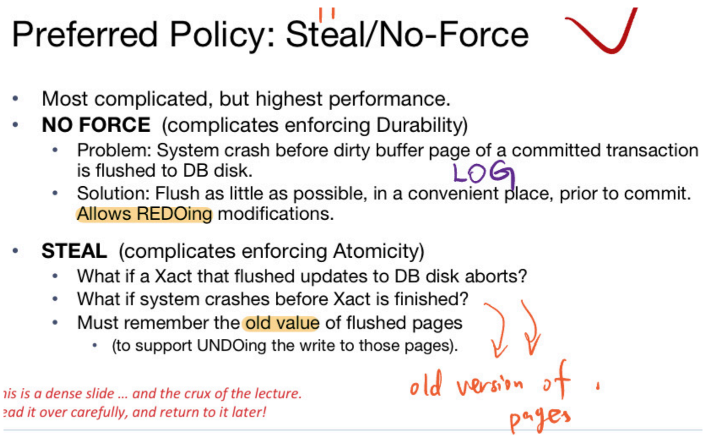


Assign priorities based on age: (now – start_time).
Say Ti wants a lock that Tj holds. Two possible policies:

- **Wait-Die:** If Ti has higher priority, Ti waits for Tj; else Ti aborts
- **Wound-Wait:** If Ti has higher priority, Tj aborts; else Ti waits
- Read each of these like a ternary operator (C/C++/java/javascript)

2 Recovery(30 pts)

The database has just crashed owing to the operator error, and this time you follow the recovery process with the **STEAL/NO FORCE** policy. You boot the database server back up, and find logging information on disk at the following tables:



Preferred Policy: Steal/No-Force

- Most complicated, but highest performance.
- **NO FORCE** (complicates enforcing Durability)
  - Problem: System crash before dirty buffer page of a committed transaction is flushed to DB disk.
  - Solution: Flush as little as possible, in a convenient place, prior to commit. Allows REDOing modifications.
- **STEAL** (complicates enforcing Atomicity)
  - What if a Xact that flushed updates to DB disk aborts?
  - What if system crashes before Xact is finished?
  - Must remember the old value of flushed pages
    - (to support UNDOing the write to those pages).

is is a dense slide … and the crux of the lecture.
ead it over carefully, and return to it later!

LOG

old version of pages

LSN	Record	prevLSN
30	update: T3 writes P5	null
40	update: T4 writes P1	null
50	update: T4 writes P5	40
60	update: T2 writes P5	null
70	update: T1 writes P2	null
80	Begin Checkpoint	-
90	update: T1 writes P3	70
100	End Checkpoint	-
110	update: T2 writes P3	60
120	T2 commit	110
130	update: T4 writes P1	50
140	T2 end	120
150	T4 abort	130
160	update: T5 writes P2	null
180	CLR: undo T4 LSN 130	150

Table 2.1: log records

Check point.

Transaction ID	LastLSN	Status
T1	70	Running
T2	60	Running
T3	30	Running
T4	50	Running

Table 2.2: Transaction Table

Page ID	RecLSN
P5	50
P1	40

Table 2.3: Dirty Page Table

1. Was the update to page 5 at LSN 60 successfully written to disk? Also, was the update to page 2 at LSN 70 successfully written to disk? Please explain both cases briefly.

① May have been flushed. it is not flushed at checkpoint but possible to be flushed later.

② Was flushed. Since the dirty page table does not contain P2

2. At the end of the Analysis phase, what transactions will be in the transaction table, and with what lastLSN and Status values? What pages will be in the dirty page table, and with what recLSN values?

first cause page to be dirty

Transaction ID	LastLSN	Status
T1	90	Running
T3	30	Running
T4	180	Aborting
T5	160	Running

Page ID	RecLSN
P1	40
P2	160
P3	90
P5	50

→ T4 abort

} T2 commit

3. At which LSN in the log should redo begin? Which log records will be redone (list their LSNs)? All other log records will be skipped.

Start from smallest LSN in dirty table: 40

40, 50, 60, (80), 90, (100), 110, (120), 130, (140), (150)

110 180.

RecLSN

70 < 160, So no 70.

(): can both be done or not.

3 Parallel Query Processing(15 pts)

We've discussed 4 kinds of parallel join in class:

- **Parallel Hash Joins**: Use hash partitioning on both relations with the same hash function, then perform a normal hash join on each machine independently.
- **Parallel Sort Merge Join**: Use range partitioning with the same ranges on both relations, then perform sort merge join on each machine independently.
- **One-sided shuffle Join**: When one relation's data is already partitioned the way we want (hash partitioned or range partitioned on a key), just partition the other relation, then run local join (using any algorithm) at every node and union results.
- **Broadcast Join**: If one relation is small, send it to all nodes that have a partition of the other relation. Do a local join at each node (using any algorithm) and union results.

In shared nothing, the machines communicate with each other solely through the network by sending data to each other. Here, network cost is referred to the amount of data sent between machines.

Now we have a Relation R that has 10,000 pages, round-robin partitioned across 4 machines (M1, M2, M3, M4). Relation S has 20 pages, all of which are only stored on M1. We want to join R and S on the condition R.col = C.col. Assume the size of each page is 1 KB.

d

1. Which type of join would have the lowest network cost in this scenario?
   a) Parallel Hash Joins
   b) Parallel Sort Merge Join
   c) One-sided shuffle Join
   d) Broadcast Join

S in small

2. How many KB of data must be sent over the network to join R and S using this join method?

Send 4 S: 20 × 4 × 1 = 80 kB.

(union cost ?)

a

3. Would the amount of data sent over the network change if R was hash or range partitioned among the 4 machines rather than round-robin partitioned using this join method?
   a) The network cost will not change under both of the partitioning methods.
   b) The network cost will change under both of the partitioning methods.
   c) The network cost will only change under range partitioning.
   d) The network cost will only change under hash partitioning.

Same of broadcast for all partition

but:

hash partitioned → parallell grace hash ✓

range partitioned → parallel sort-merge. ✓

4 DISTRIBUTED TRANSACTION(15 PTS)

1. Suppose we have one coordinator and four participants. It takes 40ms for a coordinator to send messages to all participants; 5,10,20 and 25ms for participant 1,2,3 and 4 to send a message to the coordinator respectively; and 15 ms for each machine to generate and flush a record. Assume for the same message, each participant receives it from the coordinator at the same time.
   Under proper 2PC and logging protocols, how long does the whole 2PC process (from the beginning to the coordinator's final log flush) take for a successful commit in the best case?(Please write down your analysis for the question.)

coordinator send Prepare: 40 ms.
participants generate and flush prepare: 15 ms.
max(participant send Yes to Cordinator) : 25 ms.

Cordinater generate and flush commit : 15 ms

Sum:
40+15+20+15 = 90 ms.

5 NO SQL(20 PTS)

Answer the questions about scaling, you need to give a brief analysis.

1. A small startup realizes that its current database can't sustain their growing workloads. Given that these workloads involve lot of write but few reads, should it invest in more partitioning or more replication?

partitioning, since it's easy for write (compared with replication)

2. A mechanical failure causes some of the startup's database machines to permanently crash, losing data in the process. If the startup wants to prevent similar losses in the future, should it invest more in partitioning or more replication?

replication, since it is more fault-tolerance and have mutiple replicates of data for recovering in case when some machine crashes.