University of Edinburgh
School of Informatics
INFR11199 - Advanced Database Systems (Spring 2024)

Tutorial Sheet 5

1. (Transactions & Concurrency) Consider a database with objects X and Y and two transactions. Transaction 1 reads X and Y and then writes X and Y. Transaction 2 reads and writes X then reads and writes Y.

   (a) Create a schedule for these transactions that is *not* serializable. Explain why your schedule is not serializable.

   Solution: Here is an example of a schedule that is not serializable.

Transaction 1	Transaction 2
Read(X)	
Read(Y)	
	Read(X)
	Write(X)
	Read(Y)
Write(X)	
Write(Y)	
	Write(Y)

   In this example, T1 reads X before T2 writes X. However, T1 writes X after T2 reads/writes it. The schedule is thus not serializable since there is no equivalent serial schedule that would have the same result (neither T1-T2 nor T2-T1).

   (b) Would your schedule be allowed under strict two-phase locking? Why or why not?

   Solution: No, because strict 2PL ensures serializability. Keep in mind that strict 2PL only allows releasing locks at the end of a transaction. In the example schedule shown above, when Transaction 2 attempts to

acquire an exclusive lock to write X, it will have to wait for Transaction 1 to release its lock on X, which will not happen until Transaction 1 commits. This will never happen, so this schedule is not possible under strict 2PL.

Now consider the following schedule:

	1	2	3	4	5	6	7	8
T1				X(B)	X(A)			S(D)
T2		X(D)	S(E)					
T3	X(E)						S(B)	
T4						X(A)		

(c) Draw the waits-for graph for this schedule.

Solution: The waits-for graph consists of the following edges:

- T1 $\rightarrow$ T2
- T2 $\rightarrow$ T3
- T3 $\rightarrow$ T1
- T4 $\rightarrow$ T1

(d) Are there any transactions involved in deadlock?

Solution: Yes, Transactions 1, 2, and 3 are deadlocked.

(e) Next, assume that T1 priority > T2 > T3 > T4. You are the database administrator and one of your users purchases an exclusive plan to ensure that their transaction, T2, runs to completion. Assuming the same schedule, what deadlock avoidance policy would you choose to make sure T2 commits?

Solution: Choose wait-die. Under wait-die, T3 and T4 abort after steps 6 and 7 because they are attempting to acquire a lock held by a transaction with higher priority. Afterwards, both T1 and T2 run to completion.

However, under wound-wait, T3 will be killed by T2 at step 3, and T2 will be killed by T1 at step 8. Since we want to make sure T2 commits, we should choose wait-die.

2. (Parallel Query Processing) For each of the following scenarios, state whether it is an example of:

- Inter-query parallelism
- Intra-query, inter-operator parallelism
- Intra-query, intra-operator parallelism
- No parallelism

(a) A query with a selection, followed by a projection, followed by a join, runs on a single machine with one thread.

> **Solution:** No parallelism. It might look like a pipeline, but at any given point in time there is only one thing happening, since there is only one thread.

(b) Same as before, but there is a second machine and a second query, running independently of the first machine and the first query.

> **Solution:** Inter-query parallelism.

(c) A query with a selection, followed by a projection, runs on a single machine with multiple threads; one thread is given to the selection and one thread is given to the projection.

> **Solution:** Intra-query, inter-operator parallelism.

(d) We have a single machine, and it runs recursive hash partitioning (for external hashing) with one thread.

> **Solution:** No parallelism, because there is only one machine and one thread. Don't confuse this with parallel hashing!

(e) We have a multi-machine database, and we are running a join over it. For the join, we are running parallel sort-merge join.

> **Solution:** Intra-query, intra-operator parallelism. We have a single query and a single operator, but that single operator is going to do multiple things at the same time (across different machines).

3. (Parallel Query Processing) Suppose we have 4 machines, each with 10 buffer pages. Machine 1 has a `Students` table which consists of 100 pages. Each page is 1 KB, and it takes 1 second to send 1 KB of data across the network to another machine.

(a) How long would it take to send the data over the network after we uniformly range partition the 100 pages? Assume that we can send data to multiple machines at the same time.

Solution: 25 seconds.
After we uniformly partition our data, Machine 1 will send 25 pages to Machines 2, 3, and 4. It will take 25 seconds to finish sending these pages to each machine if we send the pages to each machine at the same time.

(b) Next, imagine that there is another table, `Classes`, which is 10 pages. Using just one machine, how long would a BNLJ take if each disk access (read or write) takes 0.5 seconds?

Solution: 105 seconds.
BNLJ will require $10 + \lceil 10/8 \rceil \cdot 100 = 210$ I/Os, which takes 105 seconds.

(c) Now assume that the `Students` table has already been uniformly range partitioned across the four machines, but `Classes` is only on Machine 1. How long would a broadcast join take if we perform BNLJ on each machine? Do not worry about the cost of combining the output of the machines.

Solution: 40 seconds.
First, we must broadcast the `Classes` table to each machine, which will take 10 seconds (since we send the table to each machine at the same time). Next, we will perform BNLJ on each machine, which requires $10 + \lceil 10/8 \rceil \cdot 25 = 60$ I/Os, or 30 seconds. In total, the time required to send the data over the network and perform the join will be 40 seconds.

(d) Which algorithm performs better?

Solution: Broadcasting the `Classes` table and performing a parallel BNLJ runs faster than just using one machine, even with the additional time required to send the table over the network.

(e) Knowing that the `Students` table was range partitioned, how can we improve the performance of the join even further?

Solution: Since we know the `Students` table was range partitioned, we can also range partition the `Classes` table on the same column and only send the corresponding partitions to each machine. This should lower the network cost and decrease the number of disk I/Os.

4. (Two-Phase Commit with Logging) Suppose we have one coordinator and three participants. It takes 30ms for a coordinator to send messages to all participants; 5, 10, and 15ms for participant 1, 2, and 3 to send a message to the coordinator respectively; and 10ms for each machine to generate and flush a record. Assume that for the same message, each participant receives it from the coordinator at the same time.

   (a) Under 2PC with logging, 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?

   Solution: 130ms.
   Phase 1: 30 + 10 + 15 + 10 = 65ms
   Coordinator sends prepare message + Participant generates and flushes prepare record + max time it takes a participant to send a Yes message + Coordinator generates and flushes commit record
   Phase 2: 30 + 10 + 15 + 10 = 65ms
   Coordinator sends commit message + Participant generates and flushes commit record + max time it takes a participant to send an ACK message + Coordinator generates and flushes end record
   Total: 130ms

   (b) Now in the 2PC protocol, describe what happens if a participant receives a PREPARE message, replies with a YES vote, crashes, and restarts (All other participants also voted YES and didn't crash).

   Solution: In this scenario, upon restarting, the recovery process will find that the participant is in the prepared state for the transaction. It will periodically try to contact the coordinator site to find out how the transaction should be resolved. In our scenario, the final outcome of the transaction is a commit. Because the coordinator cannot end a committed transaction until it receives final ACKs from all nodes, it will correctly respond with a COMMIT message to the inquiry. The participant will then be able to properly commit the transaction. It will write (and flush) a COMMIT record and send an ACK message to the coordinator.

   (c) In the 2PC protocol, suppose the coordinator sends PREPARE message to Participants 1 and 2. Participant 1 sends a "VOTE YES" message, and Participant 2 sends a "VOTE NO" message back to the coordinator.

      i. Before receiving the result of the commit/abort vote from the coordinator, Participant 1 crashes. Upon recovery, what actions does Participant 1 take (1) if we were not using presumed abort, and (2) if we were using presumed abort?

Solution: Upon recovery, Participant 1 will find a PREPARE message for this transaction in their log. In both cases, Participant 1 sends an inquiry to the coordinator for the status of this transaction and receives a message that the transaction was aborted. Participant 1 will abort the transaction locally.

ii. Before receiving the result of the commit/abort vote from the coordinator, Participant 2 crashes. Upon recovery, what actions does Participant 2 take (1) if we were not using presumed abort, and (2) if we were using presumed abort?

Solution: If running 2PC without presumed abort, Participant 2 sees a (flushed) abort record for this transaction in their log and sends the "VOTE NO" message back to the coordinator.
If running 2PC with presumed abort, the abort record may not have reached disk before crashing, thus it's missing upon recovery. Participant 2 will abort the transaction locally, without sending any messages to the coordinator.

(d) In the 2PC protocol, suppose that the coordinator sends PREPARE messages to the participants and crashes before receiving any votes from the participants. Assuming that we are running 2PC with presumed abort, answer the following questions.

i. What sequence of operations does the coordinator take after it recovers?

Solution: After the coordinator restarts, the recovery process will find that a transaction was executing at the time of the crash and that no commit log record had been written (remember that the coordinator does not write any log records before sending PREPARE messages). The recovery process will abort the transaction locally (not sending an ABORT message) by undoing its actions, if any, and writing an abort record.

ii. What sequence of operations does a participant who received the message and replied NO before the coordinator crashed take?

Solution: If the participant sent a NO vote, it knows that the transaction will be aborted because a NO vote acts like a veto. The participant does not care if the coordinator crashes or not. It aborts the local effects of the transaction.

iii. What sequence of operations does a participant who received the message and replied YES before the coordinator crashed take?

Solution: If the participant sent a YES vote, it cannot make any unilateral decisions. If the participant notices the failure of the coordinator (for example by using a timeout), it hands the transaction over to the recovery process. The recovery process will find that it is in the prepared state for the transaction. It will periodically try to contact the coordinator site to find out how the transaction should be resolved. As we discussed above, after the coordinator recovers, it will abort the transaction and will answer "abort" upon receiving an inquiry message. The participant will then abort the transaction.

iv. Let's say that the coordinator instead crashes after successfully receiving votes from all participants, with all participants voting YES except for one NO vote. Assuming the coordinator sees no records for this transaction in its log after coming back online, how does this affect the answers to parts (d).i-(d).iii.?

Solution: Note: the coordinator's log may contain no information about the transaction or an ABORT record (because the abort record does not need to be flushed immediately with presumed abort). In this question, we assume the coordinator crashed without flushing its ABORT log record.
No change for part (d).i. because the coordinator likewise sees no log records for the transaction. The recovery process will abort the transaction locally, and since there is no information about the participant IDs in the log, the coordinator cannot send abort records to the participants.
No change for part (d).ii. - with presumed abort, even if the participant itself crashes after sending the NO vote, the participant will proceed with abort since that is the presumption given no log records.
No change for part (d).iii. - with presumed abort, when the coordinator comes back online and sees no information about the transaction in its log, the transaction is unilaterally aborted. This change is communicated when participants who voted YES ping the coordinator to find out how the transaction should be resolved, and the coordinator will respond with an ABORT message.
Note: the number of records written/flushed to disk with presumed abort is fewer than without presumed abort. However, with successful transactions, the number of flushed records will be the same.