CS W186 Databases - Fall 2019
Guerilla Section 4: Parallel Query Processing; Transactions and Concurrency

Sunday, November 3, 2019

1 Types of Parallelism

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

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

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

3. 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.

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

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

2 Partitioning for Parallelism

1. Suppose we have a table of size 50,000 KB, and our database has 10 machines. Each machine has 100 pages of buffer, and a page is 4 KB.

   We would like to perform parallel sorting on this table, so first, we perfectly range partition the data. Then on each machine, we run standard external sorting.

   How many passes does this external sort on each machine take?

2. Suppose we were doing parallel hash join. The first step is to partition the data across the machines, and we usually use hash partitioning to do this.

   Would range partitioning also work? What about round-robin partitioning?

3. Suppose we have a table of 1200 rows, perfectly range-partitioned across 3 machines in order.

   We just bought a 4th machine for our database, and we want to run parallel sorting using all 4 machines.

   The first step in parallel sorting is to repartition the data across all 4 machines, using range partitioning. (The new machine will get the last range.)

   For each of the first 3 machines, how many rows will it send across the network during the repartitioning? (You can assume the new ranges are also perfectly uniform.)

3 Transactions and Concurrency

In this question, we will explore the key topics of transactions and concurrency: serializability, types of locks, two-phase locking, and deadlocks.

We will do this by actually running multiple transactions at the same time on a database, and seeing what happens. You may find it helpful (but not necessary) to draw some graphs:

- For the lock type questions, you may wish to draw a graph representing the whole database and which resources are being locked.
- For serializability questions, you may wish to draw a graph with a node for each transaction, and arrows if there are *conflicts* between transactions.
- For deadlock questions, you may wish to draw a graph with a node for each transaction, and arrows if a transaction is *waiting* for a lock held by another transaction.

We will use a database with tables A, B, C, ... and table A holds rows A1, A2, A3, ... and so on.

Consider the following sequence of operations:

```
Txn 1:  IX-Lock(Database)      (1)
Txn 1:  IX-Lock(Table A)       (2)
Txn 1:  X-Lock(Row A1)         (3)
Txn 1:  Write(Row A1)          (4)
Txn 1:  Unlock(Row A1)         (5)
Txn 1:  S-Lock(Row A2)         (6)
Txn 2:  IX-Lock(Database)      (7)
Txn 2:  IX-Lock(Table A)       (8)
Txn 2:  X-Lock(Row A1)         (9)
Txn 2:  Write(Row A1)          (10)
Txn 2:  S-Lock(Row A2)         (11)
Txn 2:  Read(Row A2)           (12)
```

1. Is Transaction 1 doing Two-Phase Locking so far?

2. Is Transaction 1 doing Strict Two-Phase Locking?

3. Is this schedule conflict-serializable so far? If not, what is the cycle?

4. Is this schedule serial so far?

Continuing with all operations so far:

```
Txn 2:  SIX-Lock(Table B)        (13)
Txn 2:  X-Lock(Row B2)           (14)
Txn 2:  Write(Row B1)            (15)
Txn 2:  Unlock(Row B1)           (16)
Txn 2:  Unlock(Table B)          (17)
Txn 1:  SIX-Lock(Table B)        (18)
Txn 1:  S-Lock(Row B1)           (19)
Txn 1:  Read(Row B1)             (20)
```

5. Is Transaction 2 doing Two-Phase Locking so far?

6. Is Transaction 2 doing Strict Two-Phase Locking?

7. Is this schedule conflict-serializable so far? If not, what is the cycle?

8. Suppose we start a new transaction, Transaction 3. What kind of locks can Transaction 3 acquire on the whole database?

9. Given the above answers, what kind of locks can Transaction 3 acquire on Table A?

10. Given the above answers, what kind of locks can Transaction 3 acquire on Row A3?

11. Given the above answers, what kind of locks can Transaction 3 acquire on Table B?

12. Given the above answers, what kind of locks can Transaction 3 acquire on Row B2?

13. What kind of locks can Transaction 1 acquire on Row B2?

```
Txn 2:  IX-Lock(Table B)         (21)
Txn 3:  IX-Lock(Database)        (22)
Txn 3:  X-Lock(Table C)          (23)
Txn 3:  IX-Lock(Table A)         (24)
Txn 3:  X-Lock(Row A1)           (25)
Txn 1:  IX-Lock(Table C)         (26)
```

14. We have now entered a deadlock. What is the waits-for cycle between the transactions?

15. We can end this deadlock by aborting the youngest transaction. Which transaction do we abort?

    Alternatively, we could have avoided this deadlock in the first place by using *wound-wait* or *wait-die*. Recall from lecture that these methods cause transactions to sometimes abort, according to a priority order, when they try to acquire locks.

    Let's say that the priority of the transaction is its number (Txn 1 is highest priority).

16. If we were using *wound-wait*, what is the **first operation** in this sequence that would cause a transaction to get aborted, and which transaction gets aborted?

17. If we were using *wait-die*, what is the **first operation** in this sequence that would cause a transaction to get aborted, and which transaction gets aborted?