Data Management (A.A. 2023/24) – exam of 11/07/2024 Solutions

Problem 1 We refer to a setting with both shared and exclusive locks. A transaction T is called *cautious* if all its lock requests appear before every other action of T. A schedule is called *cautious* if it is legal and all its transactions are both well-formed and cautious. A lock-based, active scheduler with both shared and exclusive locks is called *heedful* if it behaves like a 2PL scheduler with the only addition that it does not accept schedules that are not cautious (where we adopt the usual definition of an active 2PL scheduler "accepting" a schedule). Prove or disprove the following sentences.

1.1 Every cautious schedule is a 2PL schedule.
1.2 Every cautious schedule is conflict serializable.
1.3 Every heedful sheduler can avoid to deal with deadlock management.

Solution

1.1 We remind the reader that a "2PL schedule" is a member of the class of schedules defined as follows ($DT(S)$ denotes the "data action projection of S", i.e., the projection of the schedule S onto the actions of type read, write, commit, abort):

$\{DT(S) \mid$ there exists a schedule S' such that S is the output of a 2PL scheduler with shared and exclusive locks when processing $S'\}$.

In other words, the class includes exactly those $DT(S)$ for some S generated by a 2PL scheduler with both shared and exclusive locks.

Given this definition, it is not difficult to disprove that every cautious schedule is a 2PL schedule. Indeed, it is sufficient to consider the schedule S_1:

$$sl_1(x)\, xl_1(y)\, r_1(x)\, xl_2(y)\, xl_2(x)\, w_2(x)\, w_2(y)\, w_1(y)$$

and notice that a 2PL scheduler will never output the sequence of actions S':

$$r_1(x)\, w_2(x)\, w_1(y)\, w_1(y)$$

because it will put transaction 2 in a waiting stage when such transaction asks for the exclusive lock on y.

1.2 We disprove that every cautious schedule is conflict serializable by using the schedule S_1 above, which is cautious, but not conflict serializable.

1.3 We prove that every heedful sheduler can avoid to deal with deadlock management by observing that for a schedule S procesessed by a heedful schedule, if a transaction T in S is waiting for the release of a lock by another transaction, then T has executed no action yet and therefore the node corresponding to T in the wait for graph corresponding to S has no incoming edge. On the other hand, if T has executed at least an action, then it has obtained all the locks it needs, and therefore it will never wait for a lock, implying that the node corresponding to T in the wait for graph corresponding to S has no outgoing edge. We conclude that no node in the waiting for graph corresponding to S will have outgoing edges and incomeing edges simultaneously and therefore such graph is acyclic, implying that no deadlock can occure when processing S.

Problem 2 Consider the following transactions: $\boldsymbol{T_1} : r_1(A)\, w_1(C)$; $\boldsymbol{T_2} : r_2(B)\, w_2(A)$; $\boldsymbol{T_3} : r_3(C)\, w_3(D)$; $\boldsymbol{T_4} : w_4(D)\, r_4(B)$ and answer the following questions, providing a detailed justification for each answer.

2.1 How many non-serializable schedules do exist on T_1, T_2, T_3, T_4?
2.2 Prove or disprove that all serializable schedules on T_1, T_2, T_3, T_4 are accepted by the 2PL scheduler with both shared and exclusive locks.

Solution

2.1 We notice that the possible conflicts are only between $r_1(A)$ and $w_2(A)$, between $w_1(C)$ and $r_3(C)$, and between $w_3(D)$ and $w_4(C)$. This means that no matter how we form a schedule S on T_1, T_2, T_3, T_4, the precedence graph associated to S will have no cycle. Therefore, all the schedules that we can build on T_1, T_2, T_3, T_4 are conflict serializable, and hence serializable.

2.2 We disprove that all serializable schedules on T_1, T_2, T_3, T_4 are accepted by the 2PL scheduler with both shared and exclusive locks by exhibiting the following schedule:

$$r_1(A(\, r_2(B)\, w_2(A)\, r_3(C)\, w_1(C)\, ...$$

that is clearly not accepted by the 2PL scheduler with both shared and exclusive locks.

Problem 3 Given the table `R(A,B,C)` with 1000 tuples stored in a heap with 100 pages and given 80 frames in the buffer, we want to answer the query:

```
select A, count(*) from R group by A having count(*) > 10
```

3.1 Illustrate in detail a multi-pass algorithm to solve the problem, and tell which is its cost.

3.2 Tell whether the block-nested loop technique is suitable for answering the query.

3.2.1 If the answer is positive, then illustrate the algorithm and tell which is its cost.

3.2.2 If the answer is negative, consider the following question: can we build and use an index during the application of the block-nested loop algorithm in such a way that the modified block-nested loop algorithm is able to produce the result of the query? If the answer to the question is negative, then motivate the answer in detail; if the answer is positive, provide a detailed description of the new algorithm and tell which is its cost.

Solution

3.1 Notice that if we could assume that all values of all the attributes are of the same size, then, on the basis of the fact that 1000 tuples are stored in 100 pages, we could conclude that 30 values fit in one page, which means that all the values of attribute `A` fit in 34 pages. Hence, since we only need the values of attributes `A` to answer the query, in this case we can simply use the 1-pass algorithm with a cost of 100 page accesses.

However, there is nothing that suggest we can make the above assumption, and therefore we have to solve the problem in the worst case scenario, the one where it is not the case that all values in the attribute `A` fit in the buffer, which correspnd to the case where for each tuple of `R`, the value of `A` occupies more than 80% of the space of the whole tuple. In this case, since $100 \leq 80 \times 79$, it is immediate to notice that we could apply the 2-pass algorithm based on sorting to answer the query, with a cost of $3 \times 100 = 300$ page accesses.

3.2 Again, we refer to the general case, i.e., the case where we cannot assume that all values in the attribute `A` fit in the buffer. In this case, we remind the reader that the blaock-nested loop cannot be used to answer "group by" queries.

3.2.2 We now show that we can indeed modify the block-nested loop algorithm by adding the usage of an index so as to produce the result of the query. The idea is to loop on all the pages of the relation, but at the same time to create and manage a hash-based index on `A`. More specifically, after loading a page P of the relation `R`, for each tuple t in P, we consider the value $t.A$ and we do the following:

* if $t.A$ is not in the index, then we insert $t.A$ in the index itself and we associate to this entry an integer value used to count the number of tuples with the value $t.A$ in `A`; we initialize this entry with the number of tuples in P with $t.A$ in `A`.
* if $t.A$ is already in the index, we increment the integer associated to it.

After the analysis of all the pages of R, the index itself will contain the result to the query.

Assuming 1 to be the cost of accessing the index for reading or writing, the cost of the algorithm is $100 + 1000 = 1.100$.

Problem 4 We refer to a setting where the data manager has 3 buffer frames available, we have a table R(A,B,C,D) with 4.000 tuples stored in a heap with 200 pages and we want to compute the relation obtained from R by eliminating duplicates.

4.1 Illustrate in detail a multipass algorithm for solving the problem in the above mentioned setting and tell which is its cost.

4.2 Making use of a hash-based index, design an index-based algorithm for solving the problem in the above mentioned setting, provide a detailed description of such algorithm and tell which is its cost.

4.3 Making use of a B^*-tree-based index, design an index-based algorithm for solving the problem in the above mentioned setting, provide a detailed description of such algorithm and tell which is its cost.

Solution

If we use a multipass algorithm, we can use a classical multipass algorithm based on sorting, that in this case requires $\log_2 200 = 8$ passes for producing a number of sorted sublists less than 3 and then one pass for eliminating the duplicates by merging. So the cost is $2 \times 8 \times 200 + 200 = 3.400$ page accesses.

If we have to use an index-based algorithm using a hash index, we can choose a good has function on the set of all the attributes and initialize an empty hash index whose search key is the set of all attributes, scan the relation using one page at a time, and then analyzing every tuple t of R doing the following:

- if t is not in the index, then we insert t in the index itself and we copy the tuple in the output frame (managed as usual);
- if t is already in the index, then we ignore it.

Assuming 1 to be the cost of accessing the index for reading or writing, the cost of the algorithm is $200 + 4.000 = 4.200$ page accesses.

If we have to use an index-based algorithm using a tree-based index, we can realize an algorithm that simply builds an ISAM index with search key the set of all attributes, avoding duplicates in the data entries. At the end, the result will be in the leaves. Assuming that all the attributes and pointer have the same size, we know that 80 values fit in one page, and therefore 16 index antries fit in one page. Therefore, the fan out of the ISAM tree is 16. The worst case is when there are no duplicates and in this case we will have 400 leaves. The height of the tree is $\log_{16} 200 = 2$. The cost is therefore $200 + 2 \times 4.000 = 8.200$.

Problem 5 (only for students who do **not** do the project)
Consider the relations Drone(num,model,date) with 1.000 pages and 30.000 tuples, and Mission(code,num,model,date) with 4.000 pages and an associated index on Mission with search key ⟨model,date⟩, for which we know that the cost of retrieving the records with a specific value of attribute model is 6 page accesses. Assume a buffer with 100 frames, and consider the two queries shown below.

Query Q_1:

```
select num, model from Drone
except all  -- not removing duplicates
select num, model from Mission
```

Query Q_2:

```
select model, date from Drone
except all  -- not removing duplicates
select model, date from Mission
```

where "except all" denotes bag difference. For both queries Q_1 and Q_2, tell (i) whether it is possible to process the query by using a block-nested loop algorithm, and (ii) whether it is possible to process the query by an index-based algorithm using the above-mentioned index on Mission. In all four cases, if the answer is positive, then describe the algorithm and tell which is its cost. If the answer is negative, then motivate the answer in detail.