CS3223 Tutorial 5: Join Algorithms

Wk 7, Sem 2, 2022/23

Questions to be discussed: 2 and 3.

1. (Exercise 14.4 R&G) This question is a continuation of tutorial 4's question 2.

   Consider the join $R \bowtie_{R.a=S.b} S$, given the following information about the relations to be joined. The cost metric is the number of page I/Os unless otherwise noted, and the cost of writing out the result should be uniformly ignored.

   - Relation R contains 10,000 tuples and has 10 tuples per page.
   - Relation S contains 2000 tuples and also has 10 tuples per page.
   - Attribute b of relation S is the primary key for S.
   - Both relations are stored as simple heap files.
   - Neither relation has any indexes built on it.
   - 52 buffer pages are available.

   (a) What is the cost of joining R and S using a **sort-merge join** (use the optimized variant whenever it's applicable)? What is the minimum number of buffer pages required for this cost to remain unchanged?

   (b) What is the cost of joining R and S using the **Grace hash join**? What is the minimum number of buffer pages required for this cost to remain unchanged?

Solution: We have $|R| = \frac{10000}{10} = 1000$ pages, $|S| = \frac{2000}{10} = 200$ pages, and B = 52.

(a) Since the number of initial sorted runs for the larger relation R is $\lceil \frac{|R|}{B} \rceil = 20$, the number of initial sorted runs for the smaller relation S is at most 20. Therefore, there is sufficient buffer pages to apply the optimized variant of sort-merge join without any merging passes for the sorting. By performing the optimized sort-merge join with R as outer relation, each outer tuple will join with at most one inner tuple (since attribute b is the primary key of S); thus, the join does not require any re-scanning of the inner tuples. Therefore, the I/O cost $= 3 \times (|R| + |S|) = 3{,}600$. To use the optimized variant of sort-merge join without any merging passes, the following inequality must be satisfied:

$$B \geq \lceil \frac{|R|}{B} \rceil + \lceil \frac{|S|}{B} \rceil + 1$$

Approximately, $B \geq \sqrt{|R| + |S|} \approx 35$. Since B=36 satisfies the above inequality but B=35 does not, the minimum number of buffer pages required is 36.

(b) Recall that the cost of Hash Join is $3 \times (|R| + |S|)$ if uniform partitioning of our hash function is assumed and $B > \sqrt{f \times |S|}$, where f is a fudge factor used to capture the small increase in size involved in building a hash table, and $|S|$ is the number of pages in the smaller relation S which is used as the build relation. Since $\sqrt{|S|} \approx 14$, we can assume that this condition is met. Therefore, the total cost $= 3 \times (|R| + |S|) = 3{,}600$. The minimum number of buffer pages required depends on f.

2. (Exercise 14.5 R&G) This question is a continuation of tutorial 4's question 3.

   Consider the join $R \bowtie_{R.a=S.b} S$, given the following information about the relations to be joined. The cost metric is the number of page I/Os unless otherwise noted, and the cost of writing out the result should be uniformly ignored.

   - Relation R contains 10,000 tuples and has 10 tuples per page.
   - Relation S contains 2000 tuples and also has 10 tuples per page.
   - Attribute b of relation S is the primary key for S.
   - Both relations are stored as simple heap files.
   - Neither relation has any indexes built on it.
   - 52 buffer pages are available.
   - Assume that any B$^+$-tree index on R has two levels of internal nodes, and any B$^+$-tree index on S has one level of internal nodes.

   (a) If only 15 buffer pages were available, what would be the cost of a **sort-merge join** (use the optimized variant whenever it's applicable)? What would be the cost of a **Grace hash join**?

   (b) If the size of S were increased to also be 10,000 tuples, but only 15 buffer pages were available, what would be the cost of a **sort-merge join** (use the optimized variant whenever it's applicable)? What would be the cost of a **Grace hash join**?

   (c) If the size of S were increased to also be 10,000 tuples, and 52 buffer pages were available, what would be the cost of a **sort-merge join** (use the optimized variant whenever it's applicable)? What would be the cost of a **Grace hash join**?

Solution:

For sort-merge join, one issue to consider is whether parts of the inner relation need to be re-scanned and if so, how this will increase the I/O cost of the join computation. Since the join column is the primary key of S, if S was chosen as the outer relation, there will be no re-scanning of the inner relation R. In general, for sort-merge join, the outer relation should be chosen to minimize the extent of the re-scanning of the inner relation.

(a) **Sort-Merge Join**. With 15 buffer pages, the number of initial sorted runs for R and S, are, respectively, $\lceil \frac{1000}{15} \rceil = 67$ and $\lceil \frac{200}{15} \rceil = 14$. Therefore, using simple sort-merge join, R and S can be sorted in a total of $1+\log_{14}(67) = 3$ and $1+\log_{14}(14) = 2$ passes, respectively. The costs for sorting R and S, respectively, are $2 \times 3 \times |R| = 6{,}000$ and $2 \times 2 \times |S| = 800$. The cost of the merging phase is $|R| + |S| = 1{,}200$. Therefore, the total I/O cost for the simple sort-merge join = 6,000 + 800 + 1,200 = 8,000.

However, the performance can be improved using the optimized sort-merge join. Observe that after one pass of merging sorted runs, the number of sorted runs in R and S become 5 and 1, respectively. Since 15 buffer pages are sufficient to merge and join these 6 sorted runs simultaneously, it is not necessary to have a second pass of merging sorted runs of R and S. Therefore, for the optimized sort-merge join, the cost for generating the initial sorted runs is $2(|R| + |S|)$, the cost for one pass of merging the initial sorted runs is $2(|R| + |S|)$, and the cost for merging and joining the partially merged sorted runs is $|R| + |S|$. The total cost is $5(|R| + |S|) = 5(1000 + 200) = 6000$.

Hash Join. We use the smaller relation, S, as the build relation. With 15 buffer pages, the first scan of S partitions it into $B-1 = 14$ buckets, each containing about $\lceil \frac{200}{14} \rceil = 15$ pages assuming a uniform hash partitioning. Therefore, the hash table for each build partition would require $f \times 15$ pages, which is more than B. We must apply the Hash Join technique again to all partitions of R and S that were created by the first partitioning phase. Then we can fit the hash table for each build partition of S in memory. The total cost will be the cost of two partitioning phases plus the cost of one probing phase. The total cost = $2 \times (2 \times (|R| + |S|)) + (|R| + |S|) = 6{,}000$.

(b) Now, $||S|| = 10,000$ and $|S| = \lceil \frac{10,000}{10} \rceil = 1,000$.

Sort-Merge Join. Since $|S| = |R|$, with 15 buffer pages, the number of initial sorted runs for S is also 67, and S can be sorted in a total of 3 passes. The costs of sorting R and S are the same (i.e., 6000); and the merging cost is $|R| + |S| = 2000$. Therefore, the total cost of using the simple sort-merge join is $6,000 + 6,000 + 2,000 = 14,000$. Again here, the performance can be improved by using the optimized variant of sort-merge join with just a single pass of merging the sorted runs. The I/O cost for the optimized sort-merge join is $5(|R| + |S|) = 10,000$.

Hash Join. Now both relations are the same size, so we can treat either one as the build relation. Assume that S is used as the build relation. With 15 buffer pages, the first scan of S partitions it into 14 buckets, each containing about $\lceil \frac{1000}{14} \rceil = 72$ pages. So again, we have to deal with partition overflow. We must apply the Hash Join technique again to all partitions of R and S that were created by the first partitioning phase. After repartitioning, the size of each partition becomes $\lceil \frac{72}{14} \rceil = 6$ pages. Now we can fit the hash table for each build partition in memory. The total I/O cost will be the cost of two partioning phases plus the cost of one probing phase. The total cost $= 2 \times (2 \times (|R| + |S|)) + (|R| + |S|) = 10,000$.

(c) Now, $||S|| = 10,000$, $|S| = \lceil \frac{10,000}{10} \rceil = 1,000$, and $B = 52$.

Sort-Merge Join. With 52 buffer pages, the number of initial sorted runs created for each relation is $\lceil \frac{1000}{52} \rceil = 20$. Therefore, using the optimized sort-merge join algorithm, we can merge and join the initial sorted runs in a single pass. The total cost is $3 \times (|R| + |S|) = 3 \times (1,000 + 1,000) = 6{,}000$.

Hash Join. Now both relations are the same size, so we can treat either one as the build relation. Assume that S is the build relation. With 52 buffer pages, the first scan of S partitions it into 51 buckets, each containing about $\lceil \frac{1000}{51} \rceil = 20$ pages. This time we do not have to deal with partition overflow. The total cost will be the cost of one partitioning phase plus the cost of one probing phase. The total I/O cost $= (2 \times (|R| + |S|)) + (|R| + |S|) = 6{,}000$.

3. (Exercise 15.9, R&G) Consider the following scenario:

   - Emp (eid: integer, sal: integer, age: real, did: integer)
   - Dept (did: integer, projid: integer, budget: real, status: char(10))
   - Proj (projid: integer, code: integer, report: varchar)

   Assume that each Emp record is 20 bytes long, each Dept record is 40 bytes long, and each Proj record is 2000 bytes long on average. There are 20,000 tuples in Emp, 5000 tuples in Dept (note that did is not a key), and 1000 tuples in Proj. Each department, identified by did, has 10 projects on average. The file system supports 4000 byte pages, and 12 buffer pages are available. All following questions are based on this information. You can assume uniform distribution of values. State any additional assumptions. The cost metric to use is the number of page I/Os. Ignore the cost of writing out the final result.

   In this question, consider the following join algorithms: block nested loop join, indexed nested-loop join, sort merge join (use the optimized variant whenever applicable), and Grace hash join.

   Consider the following query: SELECT * FROM Emp E, Dept D WHERE E.did=D.did

   (a) Suppose that there is a format-1 hash index on Emp.did. List all the plans that are considered and identify the plan with the lowest estimated cost.

   (b) Assume that both relations are sorted on the join column. List all the plans that are considered and show the plan with the lowest estimated cost.

   (c) Suppose that there is a format-1 B+-tree index on Emp.did and Dept is sorted on did. List all the plans that are considered and identify the plan with the lowest estimated cost.

Solution:

- Each page can hold $\frac{4000}{20} = 200$ Emp tuples. $|Emp| = \frac{20{,}000}{200} = 100$.
- Each page can hold $\frac{4000}{40} = 100$ Dept tuples. $|Dept| = \frac{5000}{100} = 50$.

(a) There are 7 possible plans.

P1. Block nested loop join with Emp as inner relation.

P2. Block nested loop join with Emp as outer relation.

P3. Index nested-loop join with Emp as inner relation.

P4. Sort merge join with Emp as outer relation.

P5. Sort merge join with Emp as inner relation.

P6. Hash join with Emp as probe relation.

P7. Hash join with Emp as build relation.

Let Cost(Pi) denote the cost of plan Pi.

Cost(P1) $= 50 + (\lceil \frac{50}{12-2} \rceil \times 100) = 550$.

Since $|Emp| > |Dept|$, Cost(P2) $\geq$ Cost(P1).

Cost(P3) $= 50 + (5000 \times J)$, where J is the number of index pages accessed per outer tuple. Since $J \geq 1$, Cost(P3) is at least 5050, and P3 can't be the optimal query plan.

Consider plan P4. Number of initial sorted runs for Emp $= \lceil \frac{100}{12} \rceil = 9$. Number of initial sorted runs for Dept $= \lceil \frac{50}{12} \rceil = 5$. Using the optimized variant of sort-merge join, we perform one merging pass for Dept to obtain a single sorted run, and then merge and join Dept and Emp. Therefore, Cost(P4) = (cost of pass 0 for Dept) + (cost of pass 1 for Dept) + (cost of pass 0 for Emp) + (cost of merging & joining Dept and Emp) $= (2 \times 50) + (2 \times 50) + (2 \times 100) + (50 + 100) = 550$. Note that for the partial sorting of the join operands here, it is sufficient to perform one pass of merging for either Dept or Emp; since $|Dept| < |Emp|$, it is cheaper to do so for the smaller operand. For P4, we assume

that any re-scanning of Dept does not incur additional I/O cost: since each department has 10 projects on the average, each outer Emp tuple is expected to join with 10 inner Dept tuples (which can fit onto a single page or span at most two pages); i.e., any rescanned Dept page is likely to be still resident in the buffer pool.

For P5, each outer Dept tuple is expected to join with 40 inner Emp tuples (assuming that there are $\frac{5000}{10} = 500$ distinct departments each with $\frac{20000}{500} = 40$ employees) which can fit onto a single page (or span at most 2 pages); we similarly assume that any rescanned Emp page is likely to be still resident in the buffer pool and does not incur additional I/O cost. Thus, Cost(P5) = Cost(P4).

Consider plan P6. Since the size of each Dept partition is $\lceil \frac{50}{11} \rceil = 5$ pages; we assume that there's no partition overflow. Therefore, Cost(P6) = 3(100 + 50) = 450.

Since $|Emp| > |Dept|$, Cost(P7) $\geq$ Cost(P6).

Therefore, P6 is the cheapest plan.

(b) There are 6 possible plans: P1, P2, P4, P5, P6, and P7. The costs of these plans are the same as those in part (a) except for P4 and P5. Since both the relations are already sorted on the join column, plans P4 and P5 do not need to sort the join operands before merging them to compute the join. Cost(P4) = Cost(P5) = 50 + 100 = 150. Therefore sort-merge join is the cheapest plan.

(c) There are 7 possible plans: P1, P2, P3, P4, P5, P6, and P7. The costs of these plans are the same as those in part (a) except for P3, P4 and P5. Cost(P4) and Cost(P5) are the same as those in part (b) since the join operands are already sorted. Cost(P3) = $50 + (5000 \times J')$, and since $J' \geq 1$, $Cost(P3) \geq 5050$. Therefore, the sort-merge join is the cheapest plan.