Database Tuning, ITU, Spring 2008

Rasmus Pagh

March 12, 2008

1 Optimizing join order

Suppose we want to compute the natural join of relations $R_1(a, b)$, $R_2(b, c)$ and $R_3(c, d)$. All attributes are of type `int`. There are three possible join orders:

$(R_1 \bowtie R_2) \bowtie R_3$, $(R_1 \bowtie R_3) \bowtie R_2$, and $(R_2 \bowtie R_3) \bowtie R_1$.

We seek to find the join order that minimizes the size of the intermediate result. The database has kept "statistics", based on which we should make size estimates of each of the possible intermediate results. First of all, the relation sizes are:

$|R_1| = 10,000$, $|R_2| = 50,000$, $|R_3| = 2,000$.

Furthermore, the system maintains 3 independent *tug-of-war* signatures for each attribute in the relations. This is the special case of Fast-AGMS where only one difference is computed. The signatures are shown in the following table:

	$R_1(a)$	$R_1(b)$	$R_2(b)$	$R_2(c)$	$R_3(c)$	$R_3(d)$
Signature 1	90	900	1000	500	1800	110
Signature 2	-80	-1000	-1100	700	1500	-100
Signature 3	70	700	900	-900	-2200	80

Finally, the system maintains random samples R_1', R_2', R_3', consisting of 1% of the tuples in R_1, R_2, and R_3, respectively. Since the samples are small, and kept in internal memory, their join sizes can quickly be computed:

$|R_1' \bowtie R_2'| = 100$, $|R_2' \bowtie R_3'| = 200$, $|R_1' \bowtie R_3'| = 2,000$.

a) Compute estimates for $|R_1 \bowtie R_2|$ and $|R_2 \bowtie R_3|$ based on each of the three tug-of-war signatures. Why are the tug-of-war signatures not useful for estimating the size of $|R_1 \bowtie R_3|$?

b) Compute estimates for $|R_1 \bowtie R_2|$, $|R_2 \bowtie R_3|$ and $|R_1 \bowtie R_3|$ based on the join sizes of the sampled relations. Supposing that an `int` is 4 bytes, what are the estimated sizes of the three possible intermediate results?

c) Suppose that all binary joins are computed using two-phase sort join, as described in RG, and that no pipelining is done.

- Argue that when joining any number of relations, it is always best to choose the join order that minimizes the total size of the intermediate results.
- Give an example where computing the join of 4 relations using a left-deep join tree does not minimize the total size of intermediate results.

2 Query plans

Consider relations `R(a,b)` and `S(a,c)`, where all attributes are of type integer. Assume that both relations have N tuples and are much larger than the capacity of main memory. The SQL query

```
SELECT R.a FROM R, S WHERE S.c > 10 AND R.a = S.a;
```

translates into the relational algebra expression

$$\pi_{R.a}(\sigma_{c>10}(R \bowtie S)) \ .$$

a) Suggest an alternative, equivalent relational algebra expression that is likely to result in a faster execution plan, and should never give a worse execution plan. Argue why this is the case.

Next, consider the following query:

```
SELECT R.a
FROM R
WHERE NOT EXISTS (SELECT S.a FROM S WHERE S.c = 10);
```

Since the subquery is correlated, a DBMS may execute it for every tuple of `R`. An alternative to the above query is the following, equivalent SQL statement:

```
(SELECT R.a FROM R) EXCEPT (SELECT S.a FROM S WHERE S.c = 10);
```

b) Compare the efficiency of the two SQL statements, assuming that the DBMS executes the former in the naïve way (computing the subquery for every tuple of `R`), and uses no indexes. State the asymptotic (big-O) complexities in terms of N and B, assuming that all sorting steps and joins can be done using 2 passes.