Lecture 12: Hash Joins and Aggregation

15-445/645 Database Systems (Fall 2017)
Carnegie Mellon University
Prof. Andy Pavlo

Joins

1. For a tuple $r \in R$ and a tuple $s \in S$ that match on join attributes, concatenate r and s together into a new tuple.

2. In reality, contents of tuples can vary, and depend on the processing model, storage model, and the query

3. Different approaches to joins:

   (a) Data approach: copy the values for the attributes in the outer and inner tables into a new table. The advantage is that future operators in the query plan never need to go back to the base tables to get more data. The disadvantage is that this requires more memory

   (b) Record Ids: only copy the join keys along with the record ids of the matching tuples. Ideal for column stores because the DBMS does not copy data that is not needed for the query. This is called **late materialization**.

Hash Join

1. If tuple $r \in R$ and a tuple $s \in S$ satisfy the join condition, then they have the same value for the join attributes

2. If that value is hashed to some value i, the R tuple has to be in r_i and the S tuple in s_i

3. .....Therefore R tuples in r_i need only to be compared with S tuples in s_i????

4. **Algorithm**

   (a) Build Phase: Scan the outer relation and populate a hash table using the hash function h_1 on the join attributes

   (b) Probe Phase: Scan the inner relation and use h_1.....

5. Keys: The attributes that the query is joining the tables on

6. Values: Varies per implementation. Depends on what the operators above the join in the query plan expect in the input

   (a) Full Tuple Approach: Avoid having to retrieve outer relation tuple contents on match, but uses more memory

(b) Tuple Identifier Approach: Ideal for column stores because the DBMS doesn't fetch data from disk it doesn't need. Also better if join selectivity is low

Grace Hash Join

1. When tables don't fit on main memory, you don't want the buffer pool manager constantly swapping tables in and out
2. Name comes from **GRACE** database machine in Japan
3. **Algorithm**
   (a) Build Phase: Hash **both** tables on the join attribute into partitions using the same hash function
   (b) Probe Phase: .....
4. If the buckets don't fit in memory, then use **recursive partitioning**
5. Cost of hash join is $3(M + N)$
   (a) Partitioning phase $2(M + N)$
   (b) Probing Phase: $M + N$

Observations on Hash Joins

1. If the DBMS knows the size of the outer table, the join can use a static hash table
2. If it doesn't know the size, the the join has to use a dynamic hash table or allow for overflow pages
3. INSERT JOIN ALGORITHM COST COMPARISON TABLE

Aggregations

1. Collapse multiple tuples into a single scalar value
2. We have to choices, **Sorting** and **Hashing**
3. **GROUP BY** are usually done using hashing
4. With a hash table, the aggregation is done when inserting into hash table
5. Choice of sorting vs hashing is subtle and depends on optimizations done in each case
6. In the hash table, the value is a Group By key and a running value that is dependent on which aggregate is done

Conclusion

1. Hashing is almost always going to be better than sorting for operator execution
2. However, sorting is better on non-uniform data or when the result needs to be sorted
3. Good DBMSs uses either or both