Technical Perspective: From Binary Join to Free Join

Thomas Neumann
TUM
neumann@in.tum.de

Most queries access data from more than one relation, which makes joins between relations an extremely common operation. In many cases the execution time of a query is dominated by the processing of the involved joins. This observation has led to a wide range of techniques to speed up join processing like, e.g. efficient hash joins, bitmap filters to eliminate non-joining tuples early on, blocked lookups to hide cache latencies, and many others.

But even the most careful engineering technique cannot hide the fact that a join is fundamentally a growing operation, at least in the general case. This problem is not immediately visible because many real-world joins are foreign key / primary key joins, which means that we will have at most one join partner per foreign key, and thus the result of the join will not be larger than the inputs. But that is not always the case. In general, a join is a n:m combination, which means that the result of a join can be as large as $O(n^2)$, where n is the input size. When n is large, as it is often case for databases, this can lead to disastrous execution times.

An interesting observation in this context is that a $O(n^2)$ (intermediate) result is usually a mistake. Very few users will ask queries where the result consists of millions or billions of tuples. Usually there will be subsequent operations that eliminate most of these intermediate results until only the query result remains. The query optimizer will try to find a join execution order where the intermediate result sizes are minimized, but in the presence of growing joins and often unreliable cardinality estimates that is not an easy task. Sometimes small changes the query lead to very different execution times, which is unfortunate for the robustness of the system.

And fundamentally, some queries with growing joins are inherently hard for binary joins, independent of join order. The classic example for that are triangle queries, for example the natural join between the sets $R(a,b), S(b,c), T(c,a)$. These queries often have very large intermediate result but only small query results. And even worse, it can be shown that for some data sets any strategy that uses binary joins will have $O(n^2)$ runtime, which makes these queries intractable for large data sets. While the query result itself is only in $O(n^{1.5})$ even in worst case [1], and in reality it will usually be much smaller.

Another class of join algorithms, so called worst-case optimal joins (WCOJs), process joins not as a sequence of binary joins but as a sequence of attribute equivalence classes, where all relations involved in the equivalence class are processed within the step. This avoids the intermediate result explosion and makes triangle queries tractable. But they are rarely used in database systems because the constant factors in the implementation are higher and they tend to be slower than traditional binary queries for the more common case that the intermediate results do not grow dramatically [2]. But simply always using optimized binary joins is dangerous, they are faster on typical queries but can exhibit catastrophic performance for growing queries.

The next paper finds an interesting compromise between these two extremes. It introduces so called *free join* plans, which can process both multiple attributes and multiple relations in each step. This makes both traditional binary joins and the *general join* WCOJ algorithm special cases of free joins, as both can be expressed in the framework. In addition, new strategies can be expressed as free join plans that combine aspects of traditional joins and WCOJs in one execution plan.

This allows for avoiding the high constant factors of WCOJs for the parts of the query where they are not required, while still retaining the superior pruning power of WCOJs in the overall query. This is a very nice result that will hopefully lead to systems that are more robust and can process join queries predictable and reliable.

1. REFERENCES

[1] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 739–748. IEEE Computer Society, 2008.

[2] M. J. Freitag, M. Bandle, T. Schmidt, A. Kemper, and T. Neumann. Adopting worst-case optimal joins in relational database systems. *Proc. VLDB Endow.*, 13(11):1891–1904, 2020.