View Reviews

Paper ID
17

Paper Title
TreeTracker Join: Simple, Optimal, Fast

Reviewer #1

Questions

1. Overall Recommendation
Weak Reject

2. Summary of the paper and rationale for the recommendation
This paper presents the Tree Tracker Join (TTJ) algorithm which is inspired by the TreeTracker algorithm used in constraint satisfaction. The key claim is that this algorithm is significantly better than the best known linear-time join algorithm. The paper presents a performance study comparing the algorithm using the Join Order Benchmark.

The main challenge of the paper is that is presents the problem in a theoretical isolated setting. What is not clear from this paper is what is the application domain of this algorithm and how does it compare with many of the other algorithms and systems-level optimizations (e.g., vectorized operators) used an today's database systems. The paper lacks a compelling motivating use case where this algorithm will shine, and it looks more like an isolated theoretical result.

3. Novelty and innovation
Interesting ideas

4. Strong Points (list positive aspects of the paper, especially if recommending acceptance)
S1. Joins are an important component for analytics workloads, optimizing performance of joins is always beneficial in our quest to improve query performance.

S2. The paper makes a case for the proposed algorithm using the JOB to show the benefits.

5. Weak Points (list aspects that could be improved, especially if recommending rejection)

W1. The paper needs to fit into the context of modern query execution engines as well as the end-to-end picture of a query optimizer (more details below).

W2. The paper needs to compare the algorithm with many advancements in join algorithms, such as vectorization. The simple hash join is amenable to vectorized execution which is really efficient for modern CPUs. The proposed algorithm, on the other hand, is a branch-based row-at-a-time algorithm which is known to be pretty inefficient for modern processors.

W3. SQLite is not a great fit as a baseline for analytical workloads. Would be good to show how the improvements can be materialized in comparison to modern implementations such as DuckDB

6. Overall Evaluation (Comments on the paper as a whole regarding its contributions, degree of novelty, presentation, and adequacy to CIDR)

O1. The paper needs to clearly motivate the application scenario and how this algorithm fits into the end-to-end query processing system? In today's systems, if there is an n-way join, the optimizer will pick a join order, join operator, bushy vs. deep etc. It uses the data statistics to make such decisions. Is the proposal that this algorithm will replace these joins and across the board improve query performance? The paper needs to clearly motivate the problem in a practical setting and break out of its purely theoretical formulation.

O2. As mentioned earlier, modern join algorithm implementations are heavily optimized using vectors and SIMD instructions. The proposed algorithm is not amenable to that. So one needs to quantify the benefits of the efficiency of the algorithm compared to the inefficiencies introduced at the lower layers of the system (in how the processors are optimized for vectorized operations over memory-resident data).

O3. The paper needs to clearly tie the algorithm into the query optimizer on how it interplays with other join algorithms. Is this is the join algorithm to rule them all or is an optimizer join-ordering or join operator selection still relevant. If so, how does the optimizer account for this new algorithm in its search and costing.

O4. A comparison with DuckDB instead of SQLite is more interesting and relevant. Also, how sensitive is the performance depending on join order choice or whether a left deep or a bushy plan is chosen. How susceptible is the new join algorithm to optimizer errors in choosing the optimal order and operator? If the proposed algorithm is more robust, that itself is a plus, though it needs to be demonstrated appropriately.

Reviewer #2

Questions

1. Overall Recommendation
Weak Accept

2. Summary of the paper and rationale for the recommendation
The paper proposes TreeTracker Join, a new linear-time join algorithm that requires no query time preprocessing and matches or outperforms the performance of traditional binary hash join. The basic idea is straightforward: remove tuples from the tables that lead to no matches after probing the hash table. The paper presented correctness proof and performance evaluations of TreeTracker Join, showing significant improvement over Yannakakis and binary join algorithms.

3. Novelty and innovation
Interesting ideas

4. Strong Points (list positive aspects of the paper, especially if recommending acceptance)
S1. Improving join algorithms is an important goal for high-performance systems.
S2. The algorithm overall finishes in O(N) time yet produces very good results, with most of the cases in JOB outperforming binary join and Yannakakis join.
S3. The paper presented the intuition behind the algorithm well.

5. Weak Points (list aspects that could be improved, especially if recommending rejection)
The algorithm requires more complex control flow by bailing out early in the inner loops, as described as throwing and catching exceptions in the paper. This can much

complicate actual implementation where for high performance, most systems do not rely on exception handling provided by the language/runtime (e.g., C++ or Jave exceptions), but rather explicitly maintains return values and states. Doing so for TTJ can be complex by keeping many intermediate function states. It'd be good for the paper to comment on further such practical considerations.

6. Overall Evaluation (Comments on the paper as a whole regarding its contributions, degree of novelty, presentation, and adequacy to CIDR)

The paper presented the intuition of the algorithm well in earlier sections. As someone who doesn't work on the theoretical aspects of join algorithms, I find the paper a bit hard to follow as Section 2 ends and Section 3 starts. Performance evaluation is convincing overall, and I'd appreciate more discussion on practical considerations.

Reviewer #3

Questions

1. Overall Recommendation

Reject

2. Summary of the paper and rationale for the recommendation

The paper presents TreeTracker join. The main idea is to remove dangling tuples from the base relations while processing a join. In that sense it is a hybrid of a standard join and the Yanakakis-algorithm.

Though this work has its merits, I do not see a good match for CIDR here. This is a purely algorithmic paper which should be extended and then send to SIGMOD or PVLDB.

3. Novelty and innovation

Not a match for CIDR

4. Strong Points (list positive aspects of the paper, especially if recommending acceptance)

S1 interesting idea

5. Weak Points (list aspects that could be improved, especially if recommending rejection)

W1 no-systems aspect whatsoever, no match for CIDR

6. Overall Evaluation (Comments on the paper as a whole regarding its contributions, degree of novelty, presentation, and adequacy to CIDR)

Example 1.1. this notation that equality of attribute names implies an equality join condition is used in some parts of the db community, in particular in db theory, but not in all; you should restate precisely the semantics here. It would also help to show the join graph as a figure.