CS 657 - Paper Review 3

From Theory to Practice: Efficient Join Query Evaluation in a Parallel Database System

Nick Alvarez

13 April 2021

1 Clarity

The paper is presented in a clear manner. The authors present some background on the issue they are trying to solve, they propose a new solution, and test it against existing methods. They analyze the results and investigate areas in which they can and have improved their solution. The authors conclude with the practicality of their method and its intended use cases. At certain points, the paper went a bit heavy into other existing methods that were not compared, which did not seem as much of a prevalent issue in the context of the paper. However, the point that their method performs well when used as intended was expressed clearly.

For presenting a new method, the multiple query examples, comparisons, and breakdown of results served as an extremely useful way to see why each method worked. Research in this manner should be presented in a similar way.

2 Problems and Existing Solutions

In a parallel database system, the biggest bottleneck is the communication cost occurring during data reshuffling. This can be further reduced to network usage, load balancing (skew), and CPU costs. Those with intermediate results during joins can also create expensive operations. In the past, these issues were not as prevalent due to the rarity of large tables being joined or cyclic graphs. Limitations of existing join operations are now surfacing, however.

These problems are important as in the modern day, data tables are becoming increasingly large and joins costly, especially when queries are cyclic. With existing methods, these operations can take increasingly long amounts of time. Chu et al. states that cyclic queries can be highly suboptimal and will produce large intermediate joins, usually larger than the final answer. These significant overheads, not known until data was being stored and modified in new ways, reduce database system performance.

The problems presented in the paper are hard problems. Existing methods are insufficient, and, while new methods have been proposed, they too have limitations to their performance. There are unavoidable operation overheads that can only be mitigated, not eliminated. Additionally, as the paper states, the methods described are theoretical algorithms from other papers and only in some cases are practical. The "hardness" of this issue comes from taking useful theory and turning it into a practical application.

Before this paper was published, existing techniques could be categorized as those in current use and theoretical methods. Join queries traditionally use star-joins with aggregates, with certain "expected" table formats and sizes. New data sets do not necessarily follow this expectation, and thus face performance issues with traditional methods (Chu et al. 63). Theoretical methods proposed by other papers do offer performance gains in certain circumstances. The HyperCube shuffle, refined by Beame et al. offers this gain but states the optimal number of servers can be partitioned into a fractional number of servers, which is clearly not feasible.

3 Techniques and New Solutions

The proposed technique works by combining the HyperCube shuffle and their newly proposed Tributary join algorithm, based off the Leapfrog Triejoin. The HyperCube shuffle organizes p servers into k dimensions, and tuples are sent to some server based on their coordinates. The data is shuffled, sent to the right servers, and ready for the Tributary join. This uses arrays instead of binary trees, as sorting arrays is less costly than computing a binary tree. The authors further work to optimize this algorithm by a) configuring the HyperCube with the optimal amount of workers and minimal workload, via a breadth-first search of all possible configurations, and b) the optimal variable order for Tributary join by determining the minimal number of binary searches, thus represented as the cost.

The new method for join queries that the authors propose is not necessarily better than the state-of-the-art techniques, at least when applied in a general sense. For specific types of queries, like those with cycles and/or large intermediate join results, the HyperCube shuffle and Tributary join method performs the best. The authors note that their method executes eight times faster than traditional methods, using 98% less data, and cuts runtimes and CPU times by over seventy percent (Chu et al. 63). However, their method does not perform the best with every single type of query.

The authors demonstrate the performance of the HyperCube shuffle and Tributary join with multiple queries of different types. These results are compared to traditional methods, like a regular shuffle in conjunction with a binary symmetric hash join. Results are categorized into network usage, load balancing (skew), and CPU cost. In nearly every query, the author's proposed method received the lowest (highest performing) metrics for each category. In one example, Query 3, the proposed method is actually outperformed by a regular shuffle, however, the authors explain the reasoning as to why this is.

Limitations of the proposed methods are few but still affect its feasibility in certain situations. Queries that are acyclic and with few intermediate results are better suited to traditional methods. Very complex queries also cause the algorithm to use a high number of dimensions, removing the advantage of using this method. Chu et al. broadly conclude there is no one best method for queries, and there are some types of joins that are better suited for other methods. Another limitation that the authors tried to address was the optimization of p. With a non-integer number of shares, the HyperCube shuffle may use a non-optimal number of workers. They did come up with a feasible solution that adds minimal overhead.

4 Suggestions

The paper went very heavy on references to other types of algorithms. For comparisons, this was useful as one could see why some excelled where others did not. However, algorithms not used in comparisons were brought up but not referenced many other places. Additionally, the examples given in the paper were a bit too abstract, and practical use cases would always have been welcome.