Let's Rethink Join Optimization in Distributed Systems

Semih Salihoglu
Stanford University
semih@cs.stanford.edu

ABSTRACT

Distributed shared-nothing systems that process large-scale data has seen unprecedented developments over the last decade. The advent of Google's MapReduce [2] and Hadoop [3] has been followed by a series of systems with relational operators or SQL-like interfaces, such as Pig [8], Hive [10], Spark [12], SparkSQL [9], and Myria [4]. One of the core operations performed by these systems is evaluating relational joins. Along with these systems developments, there has also been very exciting progress on join algorithms both in the serial and distributed settings. However, the algorithmic progress in joins and the developments in large-scale data processing systems have not yet met. Current systems typically perform pairwise join plans, which perform well on data with primary and foreign key constraints, but are ill-suited and suboptimal for more complex sparse data that many modern applications process [7]. As new distributed data processing systems are rapidly being developed, we believe it is the right time to rethink how joins should be optimized in these systems. In this abstract, we argue that there is a promising opportunity to implement and experiment with a new set of join algorithms in distributed systems. Table 1 summarizes the algorithms we discuss and their properties.

The specific problem we consider is the evaluation a conjunctive join query Q of m relations R_1, ..., R_m on a cluster of p distributed machines. Let *IN* and *OUT* be the size of the input tables and output of Q, respectively. Let *MAX-OUT* be the maximum possible output of Q under all instances of the input tables. We characterize the performance of distributed algorithms with two parameters: (1) the number of rounds of communication required between machines; and (2) the network IO cost of the algorithm, which consists of the total communication incurred between the machines including writing the results to a distributed file system or database.

The optimal one-round join algorithm is the *Shares* algorithm, introduced by Afrati et. al. [1] in the MapReduce context. The cost of Shares can be characterized as O(*f(p)IN* + *OUT*) [1], where *f(p)* is a non-decreasing function in the level of parallelism. In other words, the cost of Shares increases as we increase the level of parallelism, where the rate of the increase depends on the specific query.

Shares' optimality as a one-round algorithm has shown that the only way to evaluate joins more efficiently in parallel is to design multi-round algorithms. The following three multi-round algorithms offer different cost guarantees for different classes of queries. We classify the queries according to their *cyclicity*. All of the three algorithms avoid any dependency on the level of parallelism and execute for a constant number of rounds, where the constant of each algorithm depends on the query.

Name	Rounds	Query Class	Network I/O Cost
Pairwise Joins	Multi	general	O(IN^{m-1} + *OUT*)
Shares [1]	One	general	O($f(p)$*IN* + *OUT*)
D-Y [5, 11]	Multi	acyclic	O(*IN* + *OUT*)
D-GJ [5, 7]	Multi	general	O(*IN* + *MAX-OUT*)
D-Yan-GJ [5]	Multi	treewidth-w	O(IN^{w-1} + *OUT*)

Table 1: Distributed Join Algorithms.

1. **Distributed Yannakakis (D-Y)** [5, 11]: For acyclic queries, a distributed version of the famous algorithm of Yannakakis [11] achieves the asymptotically best network I/O cost of *O(IN + OUT)*.
2. **Distributed Generic Join (D-GJ)** [5, 7]: For general queries, the distributed version of Generic Join—a recent serial algorithm with provably worst-case runtime guarantees—achieves the worst-case optimal network I/O cost of *O(IN + MAX-OUT)*.
3. **Hybrid Distributed Yannakakis and Generic Join (D-Yan-GJ)** [5]: For queries with bounded treewidth of w [5], i.e., bounded degrees of cyclicity, a hybrid version of D-Y and D-GJ can achieve a network I/O cost of O(IN^{w} + *OUT*). This can be considered a middle point between the guarantees of D-Y, and D-GJ.

In contrast, pairwise join plans can generate unnecessarily large intermediate outputs, as large as IN^{m-1}, which is asymptotically suboptimal to the algorithms we have discussed, including Shares.

We believe the developments in the theory of distributed join algorithms suggest that we should next implement and experiment with these algorithms in existing systems. In particular, we should compare their performances against each other and the join plans of existing systems under four different parameters: (1) query classes; (2) parallelism scales; (3) input skewness; and (4) execution environments, e.g., Hadoop-like disk-based systems, Spark-like systems with caching features, or Naiad-like [6] systems with streaming capacities. These parameters are very rich and pose opportunities for multiple groups to do valuable systems research. We welcome the systems research of other groups on distributed join algorithms and hope that some of these algorithms can enter the optimizers of existing distributed systems, all told making a significant contribution to a very fundamental problem that has been at the core of database research.

7th Biennial Conference on Innovative Data Systems Research (CIDR '15) January 4-7, 2015, Asilomar, California, USA..

1. REFERENCES

[1] F. N. Afrati and J. D. Ullman. Optimizing Multiway Joins in a Map-Reduce Environment. *IEEE Trans. Knowl. Data Eng.*, 23(9), 2011.

[2] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the Symposium on Operating System Design and Implementation*, 2004.

[3] Apache Hadoop. http://hadoop.apache.org/.

[4] D. Halperin, V. Teixeira de Almeida, L. L. Choo, S. Chu, P. Koutris, D. Moritz, J. Ortiz, V. Ruamviboonsuk, J. Wang, A. Whitaker, S. Xu, M. Balazinska, B. Howe, and D. Suciu. Demonstration of the Myria Big Data Management Service. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2014.

[5] C. R. M. Joglekar, S. Salihoglu. D-Y. Technical report, Stanford University, August 2014. http://infolab.

[6] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi. Naiad: A Timely Dataflow System. In *Proceedings of the ACM Symposium on Operating Systems Principles*, 2013.

[7] H. Q. Ngo, C. Ré, and A. Rudra. Skew Strikes Back: New Developments in the Theory of Join Algorithms. *SIGMOD Record*, 42(4), 2014.

[8] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: A Not-So-Foreign Language for Data Processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2008.

[9] Spark SQL. https://spark.apache.org/sql/.

[10] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: A Warehousing Solution Over a Map-Reduce Framework. *Proceedings of the VLDB Endowment*, 2(2), 2009.

[11] M. Yannakakis. Algorithms for Acyclic Database Schemes. In *Proceedings of the International Conference on Very Large Data Bases*, 1981.

[12] Zaharia, M. and Chowdhury, M. and Franklin, M. J. and Shenker, S. and Stoica, I. Spark: Cluster Computing with Working Sets. In *2nd USENIX Conference on Hot Topics in Cloud Computing*, 2010.