Thesis: The Distributed Radix Join Algorithm – Efficient Distributed Join Processing on Modern Hardware

Alexander van Renen Wolf Rödiger

Technische Universität München · Boltzmannstraße 3 · D-85748 Garching
{renen, roediger}@in.tum.de

Abstract: The ever increasing volume of data in scientific as well as commercial information systems calls for scalable databases. Distributed database systems can satisfy this demand as they are scalable to higher performance requirements simply by adding more computing nodes. However, they require efficient ways to query distributed relations in order to achieve high performance.

We describe an efficient distributed algorithm for joining two large distributed relations. Our algorithm achieves a throughput of 140 M tuples per second on a cluster, consisting of 16 standard desktop machines. Furthermore, an analysis of the scalability of the algorithm is provided and it is shown that its performance on today's commodity hardware is bound by the underlying network bandwidth.

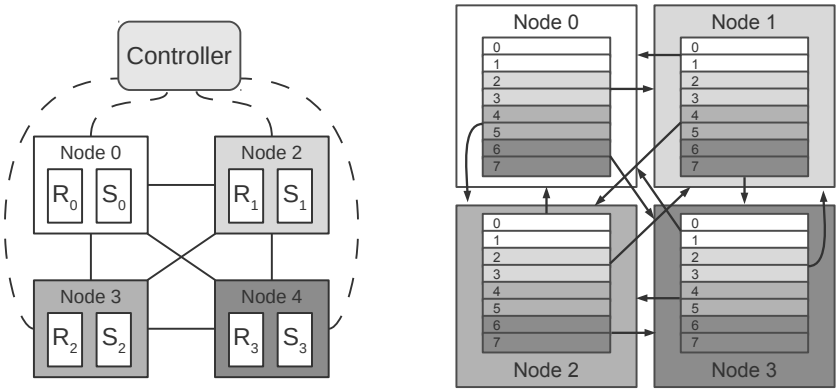
1 Motivation

Main memory database systems are one of the most important recent developments in the field of database research. Systems like HyPer [KN11] achieve unprecedented OLTP and OLAP performance which is many orders of magnitude higher than traditional disk-based database systems. However, the amount of memory which can be installed in a single server is inevitably limited. While one server might be sufficient even for large online warehouses like Amazon, some applications still exceed this limit, e.g., research facilities like CERN which need to analyze an enormous amount of data. A solution to this problem are distributed main memory database systems which allow to increase the available main memory capacity by adding more computing nodes.

The most expensive operator in distributed query processing is the join and therefore its optimization should be the primary concern when trying to achieve high query performance. This paper presents the Distributed Radix Join which adapts the centralized radix join described by Kim et al. [KKL$^+$09] to a distributed setting. Our parallel algorithm utilizes the full capabilities of today's multi-core architectures.

2 The Distributed Radix Join

The Distributed Radix Join is designed for shared-nothing [Sto86] commodity-hardware clusters, which are common in today's data centers. It computes an equi-join between two relations $\mathcal{R}$ and $\mathcal{S}$. Distributed database systems fragment the input relations across the compute nodes of a cluster. Consequently, each node holds a part $\mathcal{R}_i$ and $\mathcal{S}_i$ of both relations. We call these nodes the worker nodes as they will later perform the actual join.



(a) Fully-connected network

(b) Data distribution schema

Figure 1: The setup of the Distributed Radix Join.

An additional coordinator node synchronizes the workers. All nodes are connected to each other via a TCP network, as illustrated in Figure 1a.

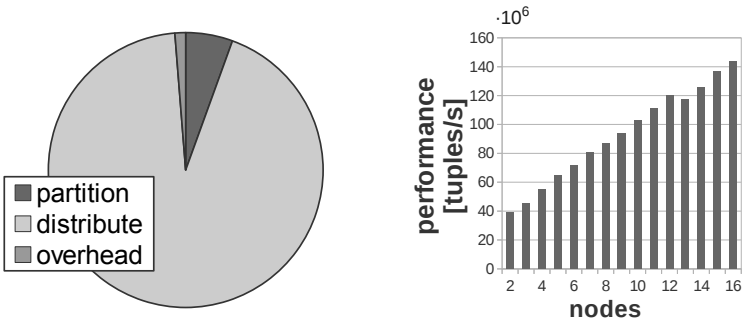
The fundamental idea of hash join algorithms is to divide the two input relations into many small disjunct partitions. As a consequence, the join can be computed independently for each partition pair [DKO+84]. We extend this idea to a distributed system, allowing a parallel join execution on separate worker nodes.

The Distributed Radix Join starts by computing the partitions on the worker nodes using the parallel radix cluster algorithm which is described in [MBK02]. Figure 1b shows an example for a radix partitioning with three bit, which results in eight partitions. After the relations are partitioned, the coordinator globally assigns the partitions to nodes, as indicated by the shades of gray. For example the white partitions 0 and 1 are assigned to node 0. In the next phase the worker nodes use the network to redistribute the currently scattered partitions to the assigned nodes. In the example, all tuples of partitions 2 and 3 are sent to node 1. The join of partition 2 is able to start as soon as all tuples of partition 2 (both R and S) are completely available at node 1.

Our implementation processes the tuples while they arrive over the network in order to both maximize the throughput and minimize the time until the first results are available. Therfore the join of two partitions of one node can be performed in parallel. The local join takes full advantage of the multi-processing capabilities of the worker node.

3 Evaluation

The Distributed Radix Join is implemented using C++11 and compiled with the open source compiler GCC-4.7. For our experiments we used a homogeneous cluster consisting of 16 computing nodes, each running a linux kernel 3.2. The nodes are equipped with an Intel Core 2 Quad Q67000 (four cores, each 2.66 GHz), 8 GB RAM main memory and a 1 GBit/s Ethernet network (in practise around 870 Mbit/s). The algorithm computes the join between two relations which each contains a 32 bit join key and a 32 bit payload.



(a) Breakdown of the execution time

(b) Scale out behavior

Figure 2: Evaluation of the Distributed Radix Join on a cluster of 16 nodes.

The centralized radix join by Kim et al. [KKL+09] achieves 100 M tuples per second on a machine equipped with an Intel Core i7 with four cores at 3.2 GHz and 6 GB RAM. Our algorithm achieves a throughput of 140 M tuples per second in our low-end cluster when joining two relations of $1600 \cdot 2^{20}$ tuples each. Figure 2a shows the breakdown of the execution time for each phase. The performance is clearly dominated by the network phase. Additionally, the local join computation utilizes only 35% of the CPU. Therefore, we expect that the join performs even better on future networks, e.g., 10 GBit Ethernet.

It is difficult to achieve the theoretical limit of the network bandwidth as several intricate effects emerge and affect the performance. An example is cross traffic which is caused by two nodes sending to the same node, thus reducing the speed of both transmissions while leaving another one unused. Our algorithm synchronizes the usage of the connections via the coordinator to avoid cross traffic and is able to achieve a throughput of 640 MBit/s on each connection. More details on these techniques can be found in [vR12].

The scale out behavior is an important measure for every distributed algorithm, as it describes how well it performs when the input and the number of nodes is increased proportionally. Figure 2b shows the performance in a cluster with up to 16 nodes on the x axis. The size of both relations is also increasing with $|\mathcal{S}| = |\mathcal{R}| = 100 \cdot 2^{20} \cdot n$, where n is the number of nodes. Within the bounds of our limited cluster size it seems as if the Distributed Radix Join exhibits a linear scale out behavior, as the performance per node is stabilizing at about 9 M tuples per second, as depicted in Figure 2b.

4 Conclusion

The Distributed Radix Join is part of our research effort to build a distributed main memory database system and will be integrated into the ScyPer system [MRR+13]. The linear scale out behavior makes it an excellent candidate for larger networks, as long as the network is fully-connected. We have shown that the join performance is bound by the network bandwidth and should therefore be able to achieve even higher performance in future networks.

Acknowledgments

Wolf Rödiger is a recipient of the Oracle External Research Fellowship, and this research is supported in part by this Oracle Fellowship.

References

[DKO+84] David J. DeWitt, Randy H. Katz, Frank Olken, Leonard D. Shapiro, Michael R. Stonebraker, and David A. Wood. Implementation techniques for main memory database systems. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, SIGMOD '84, pages 1–8, New York, NY, USA, 1984. ACM.

[KKL+09] Changkyu Kim, Tim Kaldewey, Victor W. Lee, Eric Sedlar, Anthony D. Nguyen, Nadathur Satish, Jatin Chhugani, Andrea Di Blas, and Pradeep Dubey. Sort vs. Hash revisited: fast join implementation on modern multi-core CPUs. *Proc. VLDB Endow.*, 2(2):1378–1389, August 2009.

[KN11] Alfons Kemper and Thomas Neumann. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 195–206, Washington, DC, USA, 2011. IEEE Computer Society.

[MBK02] Stefan Manegold, Peter Boncz, and Martin Kersten. Optimizing Main-Memory Join on Modern Hardware. *IEEE Trans. on Knowl. and Data Eng.*, 14(4):709–730, July 2002.

[MRR+13] Tobias Mühlbauer, Wolf Rödiger, Angelika Reiser, Alfons Kemper, and Thomas Neumann. ScyPer: A Hybrid OLTP&OLAP Distributed Main Memory Database System for Scalable Real-Time Analytics. In *GI-Fachtagung Datenbanksysteme für Business, Technologie und Web*, BTW '13, 2013.

[Sto86] M. Stonebraker. The case for shared nothing. *Database Engineering Bulletin*, 9(1):4–9, 1986.

[vR12] Alexander van Renen. Efficient Distributed Join Processing on Modern Hardware. *TUM, Bachelor's Thesis*, October 2012.