Comparison of 13 Relational Equi-joins in Main Memory

Stefan Schuh, Xiao Chen, Jens Dittrich

Presented by Áron Ricardo Perez-Lopez
Database Joins

• Relational: data presented as tables
• Join: combine tables along some criteria
• Inner join: all rows where a condition matches across the tables
• Outer join: all rows from all tables with matching ones combined
• Left join: all rows from first table extended with matching values
• Equi-join: combine based on equality of values in columns
Problems with Previous Papers

• Different implementations
• Different optimizations
• Different performance metrics $\rightarrow$ ratio of sum of relation sizes to runtime
• Different machines
• Different benchmarks
Starting Algorithms

- PRB: two-pass parallel radix join – partitioning, hash-based join
- NOP: no-partitioning hash join
- CHTJ: concise hash table join – no-partitioning hash join
- MWAY: m-way sort join – sort-merge join
Experimental Results I

Figure 1: Black box comparison of the fundamental join representatives using 32 threads and relation sizes $|R| = 128M$ and $|S| = 1280M$. 
Modified Algorithms

• (NUMA-awareness: equally allocate partition buffer over nodes)
  • Already enabled for previous tests
• PRO: modified PRB
  • Software write-combine buffers – reduces TLB misses
  • Non-temporal streaming: bypasses cache when writing – prevents polluting the cache with data that will not be read again
  • Only one pass
• PRL: PRO with linear probing instead of chaining
• PRA: PRO with dynamic array instead of hash table
• NOPA: NOP with dynamic array instead of hash table
Experimental Results II

Figure 3: Join throughput including improved versions. We observe almost a twofold performance improvement over the black-box versions shown in Figure 1.
Optimized Algorithms

• CPRL, CPRA: based on PRL and PRA
  • Original implementation causes a lot a random remote writes
  • Chunking keeps locally computed data (hash table and histogram) in the nodes and reads the data needed for joining.

• PROiS, PRLiS, PRAiS: based on PRO, PRL, and PRA
  • Original implementation causes all threads scheduled to run simultaneously to read from the same node.
  • Order of threads corresponding to different partitions changed to accommodate data available on various nodes.
Experimental Results III: PRO-derivatives

Figure 7: Runtime of PR* and CPR*-algorithms vs their variants with improved scheduling (PR*iS-algorithms). Relation sizes: $|R| = 128M, |S| = 1280M$. Lighter colors denote the partition phase and darker colors denote the join phase.
Experimental Results III: All 13 Algorithms

Figure 8: Performance of all thirteen join algorithms when using small (4 KB, dark color) and huge pages (2 MB, light color)
Experimental Results III: Scalability

Figure 10: Throughput of join algorithms when scaling input dataset sizes.
Experimental Results III: Real-World Query

Figure 14: Runtime of TPC-H Query 19, colored bars mark the fraction of the time spent in the actual join; the black bars mark the time spent for the rest of the query.
Takeaways

• Clearly specify all options used in experiments.
• Use a simple algorithm when possible.
• Be sure to make your algorithm NUMA-aware.
• Use huge pages.
• Use Software-write combine buffer.
• Be aware that join runtime ≠ query time.
• If in doubt, use a partition-based algorithm for large scale joins.
• Use the right number of partition bits for partition-based algorithms.
• Don’t use CPR* algorithms on small inputs.
Discussion Questions

• Do the authors manage to avoid the pitfalls they themselves mention at the beginning of the article?
  • Non-comparable implementations?
  • Specific machine configuration?
  • Real-world queries?

• Would it be more helpful to measure total query time?