18 – Sorting Algorithms

1. Background
    A. Process
        i. Sort phase
           sort the tuples of input relation
        ii. Merge
           scan the sorted relation, compare tuples and match the tuples.
    B. Hardware-awareness
       utilizing CPU, NUMA-aware algorithm implementation,
       using SIMD instruction
2. Parallel Sort-Merge Join
    A. Process
        i. Partition
           partition one input relation and assign them to workers
        ii. Sort
           sort the tuples of input relation/partition
        iii. Merge
           scan the sorted relation/partition,
           compare tuples and match the tuples.
    B. Partitioning
        i. Implicit partitioning
           data was partitioned on join key when it was loaded
           does not always work on OLAP environment
           in OLTP environment, most of joins are run on foreign key, so this can work.
        ii. Explicit partitioning
           divide outer relation and redistribute it among the different workers.
           similar/same with radix partitioning

C. Sort Phase

   i. Create sorted chunks of tuples for both input relations.

   ii. Can use famous sort algorithm like quicksort
   need to aware of NUMA and parallel architectures.

   iii. Cache conscious sorting
   first, in-register
   second, in cache(CPU cache : L1, L2, L3)
   last, out of order(if runs of second level exceed the size of caches)

   iv. Sorting networks
   fixed wiring paths for lists with the same number of elements
   since it does not have branch, and it has limited data dependencies, it is good on modern CPUs.

   can be used in level 1 sorting, with SIMD min/max instruction and transposing registers

   v. Bitonic merge network
   similar with sorting networks, but merging two sorted registers into single sorted run.

   vi. Multi-way merging
   using binotic merge networks, with splitting the process into tasks
   similar with merge sort with merging network.
   no synchronization, no coordination with other threads.

   real DBs do not implement this algorithm.

D. Merge phase

   i. Iterate through outer table and inner table in lockstep and compare join keys.

   ii. Can be done in parallel at the different cores without synchronization.

E. Variant : m-way, m-pass, mpsm

F. Multi-way sort merge join

   i. Sort input relations, partition it by join key, and redistribute partitions among NUMA region(only remote read/write)

   ii. Do local merge join on each NUMA region
   since it is partitioned, it does not need remote read/write
   hence this will be super fast.

G. Multi-pass sort merge join

   i. Sort input relations on locally on chunks.

   ii. Do global merge join on every pair of NUMA region

H. Massively Parallel Sort Merge join

   i. Range partition outer table, sort each partition locally by different cores

   ii. no partition on inner table, but sort each chunks locally

   iii. do merge sort globally on outer table, locally on inner table.

3. Evaluation

   A. Comparison of sort-merge joins

      i. Partition phase's performance is all same on m-way, m-pass, mpsm

      ii. Sorting phase's performance of m-way and m-pass is almost same, but mpsm's sorting phase is a lot slower.

      iii. M-way's merge phase is a lot faster than m-pass's
      also, mpsm's merge phase is slowest among the sort-merge joins

   B. Scalability of sort-merge joins

      i. M-way is good for scaling on # of threads, but, at hyper-threading area, m-way's scalability degrades.

   C. Sort-merge vs. hash

      i. Hash is better than sort-merge at all situation

      ii. Its performance gap narrows when the input relations are huge.