17 – Hash Join Algorithms

1. Background
    A. Hash Join is used in OLAP DB, since it should build hash table, and it can be big overhead when operand table is small.
    B. Hashing VS. sorting is very hot topic by some decades.
    C. Join Algorithm Design Goals
        i. Minimize Synchronization (do not overuse latch)
        ii. Minimize Memory Access Cost (awareness of NUMA/Cache/TLB)
2. Parallel Hash Join
    A. Important operator on OLAP DB
    B. Process
        i. Partition (optional)
        divide tables into sets by using hash on the join key
        ii. Build
        scan table and create hash table
        iii. Prove
        for each tuple in other table, lookup its key in hash table, if match is found, output matched tuple
    C. Partition phase
        i. Split input relations to hide cache misses.
        so, its cost should be less than cache misses of build phase
        ii. Contents of buffers depends on storage model (row / columnar)
        iii. Approach
            1. Non-blocking
                A. Shared partition
                split input relation into some chunks, hash each chunks and distribute it directly into global partitions(only one copy)

B. Private partition
split input relation into some chunks, hash each chunks(same as shared partition) and make partition locally at each chunk, and combine it into global partitions(two copy)

2. Blocking (Radix Partitioning)
scan chunk and keep counter to calculate the destination of partition. Calculating the destination index is done by prefix sum. repeat these process until target number of partitions are built.

3. Hash Functions

A. How to map a large key space into a small domain

B. Some high level hash functions

i. CRC-64 : used in networking for error detection

ii. Murmurhash : designed to a fast, general purpose hash function

iii. Google cityhash : faster for short keys

iv. Facebook XXhash : from the creator of ZSTD compression

v. Google farmhash : upgrade of cityhash with better collision rate

4. Hashing Schemes

A. How to handle key collisions after hashing

B. Approach

i. Chained hashing
used in Hyper, 64-bit bucket pointers
(48-bit pointer, 16-bit bloom filter)

ii. Linear Probe hashing
if collision occurs, use next slot. if next slot also is occupied, then see next slot.

iii. Robin Hood hashing
variant of linear probing. Steal slots from rich keys and give them to poor keys(track the number of jumps)

iv. Hopscotch hashing
variant of linear probing. Keys can move between positions in a neighborhood. Neighborhood is contiguous slots in a hash table.

v. Cuckoo hashing
use multiple tables with different hash functions.