ACM SIGMOD Programming Contest 2018

Team: PaperCup
Jaeeun Kim, Hyeonseok Oh, Kihwang Kim, Hyunsoo Cho
Hyungsoo Jung, Sooyong Kang(Advisor)
{jaeeunkim, hyeonseok, kihwangkim, hyunsoocho}@hanyang.ac.kr
{hyungsoo.jung, sykang}@hanyang.ac.kr

1. Task Overview

- The task is to evaluate batches of join queries on a set of pre-defined relations, as quickly as possible.
- Firstly, the set of relations are given in binary format. 1 sec is given to prepare for the workload.
- Next, the workload comes in batches. Each batch consists of three consecutive parts.
  1. Relations : A list of relations that will be joined.
  2. Predicates : two types of predicates are given: filter predicates and join predicates.
  3. Projections : A list of columns that are needed to compute the final check sum.

`0 2 4|0.1=1.2&1.0=2.1&0.1>3000|0.0 1.1`

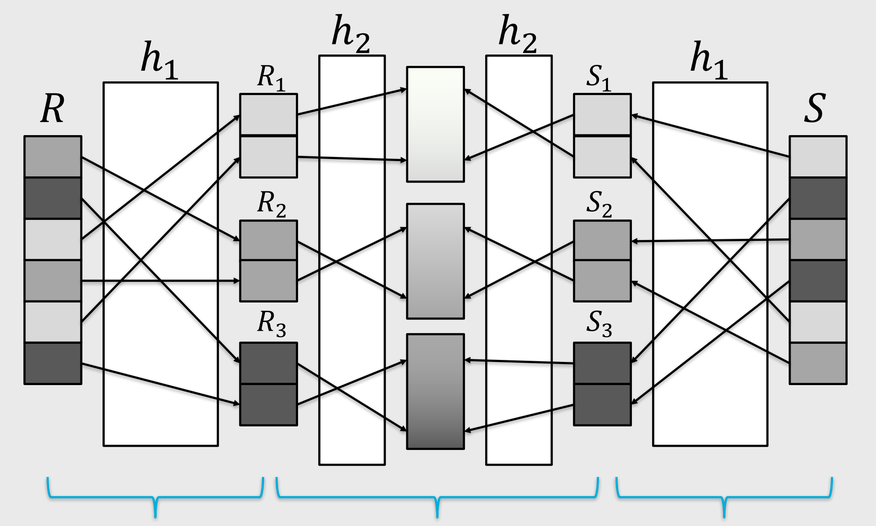
Relations | Join predicate | Filter predicate | Projections

2. Solution Overview

- We improved the performance from following aspects:
  1. Implement partitioning hash join based on “Sort vs. Hash Revisited: Fast Join Implementation on Modern Multi-Core CPUs” of Kim, Changkyu, et al. Proceedings of the VLDB Endowment 2.2 (2009): 1378-1389.
  2. Use task pool for load balancing and thread local memory pool for fast memory allocation and deallocation
  3. Propagate filter predicates to reduce rows before join
  4. Merge rows and eliminate duplicated key to reduce the number of rows to be processed
  5. Optimize the number of memory access by reducing memcpy and pointer

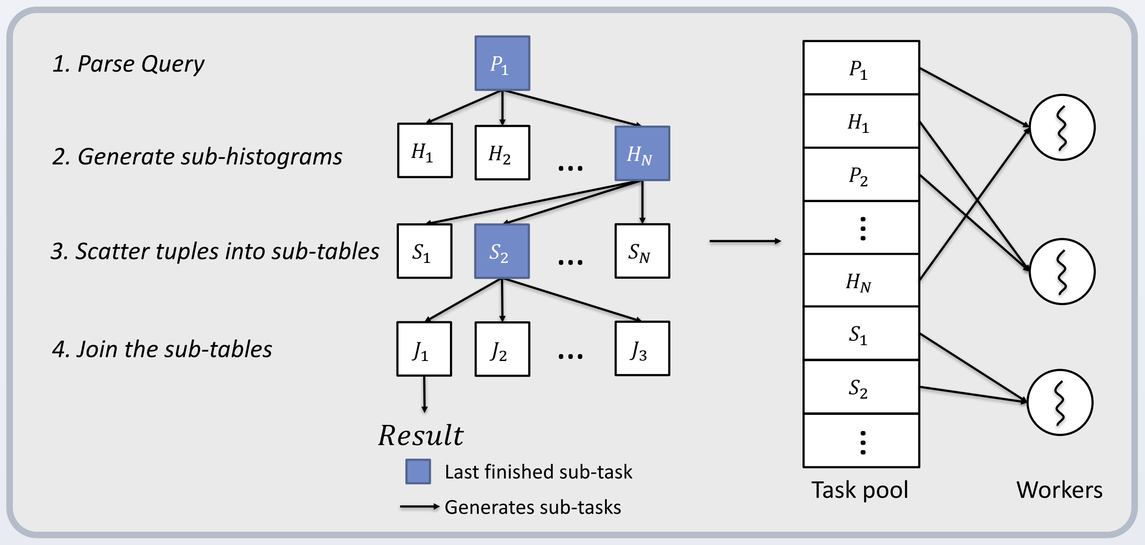
3. Partitioning Hash Join

- Join queries are processed by following order
  1. Partition data based on the rightmost *B* bits of the two tables to obtain 2^B sub-tables
  2. Build a histogram and reorder the tuples using histograms for data locality
  3. Build a hash table and probe each tuples to output the result
- We calculate *B* to partition tables into sub-tables which fit L2 cache size.



4. Task Pool & Memory Pool

- All tasks are divided into several sub-tasks and each worker thread processes sub-tasks in the task pool for load balancing
- A worker thread which processed the last sub-task for the task inserts next phase’s sub-tasks into task pool



- The main thread generates initial task and inserts into task pool from input query strings.
- After inserting all query tasks into task pool, the main thread waits until all tasks are done.
- We use thread-local memory pools to reduce space allocation overhead. Also, allocated memory space are deallocated only by the thread which allocated the space

5. Other Optimizations

Generating Query Plan

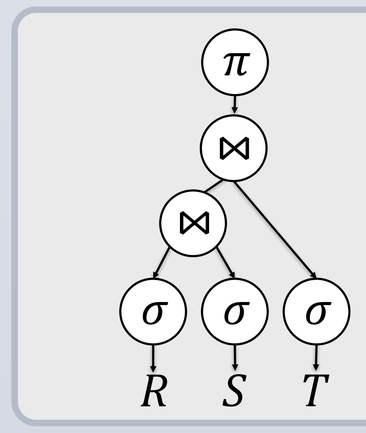
1. Firstly, propagate filter predicates which can be applied to other columns.

`R.c1=S.c2 and S.c2=T.c3 and R.c1>3`

Propagate

`R.c1=S.c2 and S.c2=T.c3 and R.c1>3 and S.c2>3 and T.c3>3`

2. Make left deep tree which always apply filter predicates first
3. Execute the query plan tree using multiple workers



Remove Duplicated rows

- If the number of columns which is requested by parent operator is 1 and we can benefit from duplicate elimination, represent each tuples as data + count

col1
10
20
10
30
20

→

col1	cnt
10	**2**
20	**2**
30	**1**

col1	cnt
10	2
20	2
30	1

$\bowtie_{col1}$

col1	col2
20	17
30	28
20	31

=

col1	col2	cnt
20	17	**2**
30	28	**1**
20	31	**2**