15-721 Project 3 Final Presentation

MULTI-THREADED QUERIES

Wendong Li (wendongl)
Lu Zhang (lzhang3)
Rui Wang (ruiw1)

Project Objective

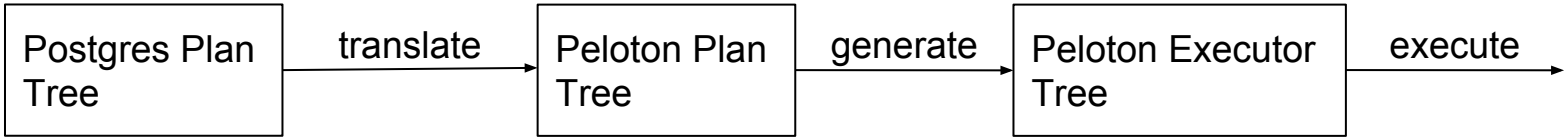
Intra-operator parallelism

- Use multiple threads in a single executor
- Focus on OLAP queries

Progress

- Insert exchange executors into the executor tree
- Concurrent data structures
- Implement different exchange executors

Progress

- Insert exchange executors into the executor tree
  - 

  - Two places to insert exchange executors
    - At translation step, translate into exchange plan node
    - At generate step, generate exchange executor node
  - We choose the second way
    - Exchange plan node does not really have anything new
    - Good for plan cache
    - Code sharing, easier if want to change plan node
    - Easier to implement

Progress

- Concurrent data structures
  - Thread pool
    - Use the one already in Peloton
  - Lock free queue
    - Implement using boost lock free queue
  - Barrier to synchronize different work threads
    - Implement using mutex and conditional variable
  - Concurrent hash table
    - Implement using Cuckoo hash map

Progress

- Implement different exchange executors
  - Decompose the entire work into multiple tasks (typically one tile group per task)
  - Use multiple threads to finish tasks
  - Wait till all tasks finish before return result
  - Implemented executors
    - Sequential scan
    - Hash
    - Hash join

Progress

- Sequential scan
  - Wrap the scanning of one tile group into one task
  - Submit multiple tasks to thread pool
  - Wait for all tasks to finish before returning result

Progress

- Hash
  - Use one concurrent hash map
  - Wrap one logical tile into one task
  - Each task is responsible for inserting one logical tile into hash table
  - Waiting for all tasks to finish before returning result

Progress

- Hash Join
  - Use Hash mentioned before to build hash table on right table
  - Use a thread-safe set to record matched records in right table if needed (left/right/outer join)

  - Two Parallel Strategies:
    - Operator-at-a-time. Configurable workload for each task. (BAD performance!)
    - Tuple-at-a-time. One tile for each task. Better, used in experiments.

Correctness

- Test under existing Peloton tests
- Create tests to test parallel part of the code

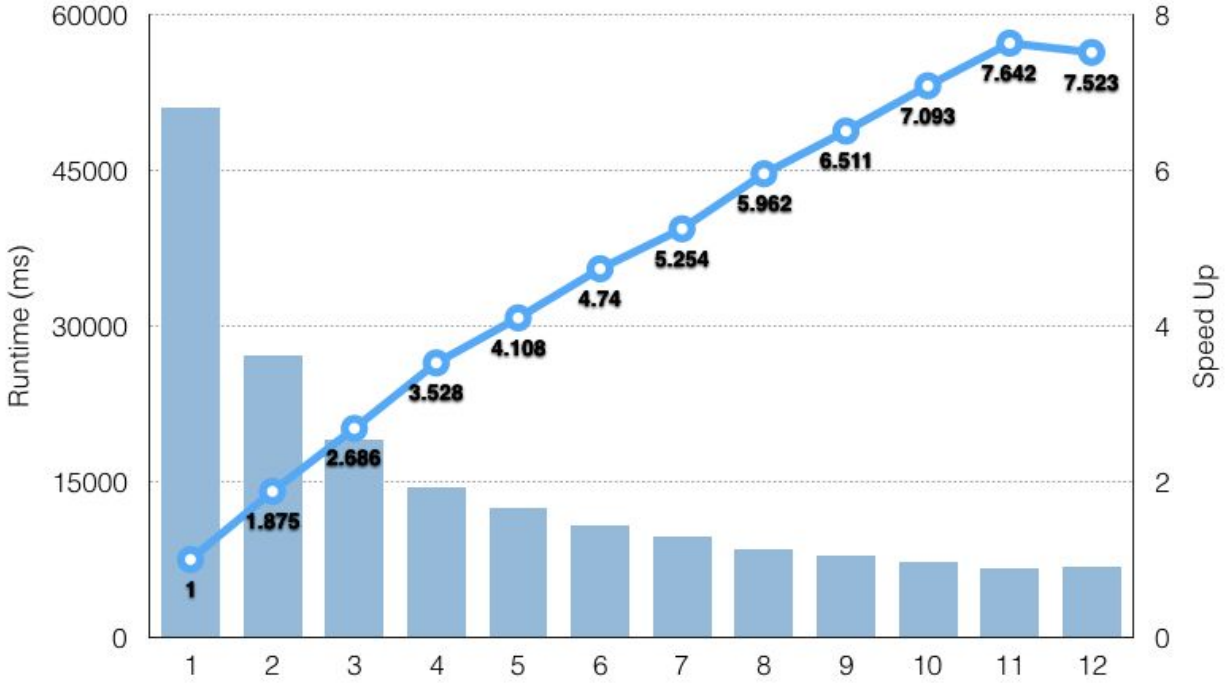
Performance

- Experiment Setup
  - MemSQL machine.
  - dual-socket Xeon E5-2620 (6 cores / 12 threads -- total of 24 threads)

- ExchangeSeqScanExecutor vs. SeqScanExecutor
- ExchangeHashExecutor vs. HashExecutor
- ExchangeHashJoinExecutor vs. HashJoinExecutor (Probe phase)

Performance: ExchangeSeqScanExecutor

- 1000 records/tile group * 100000 tile groups

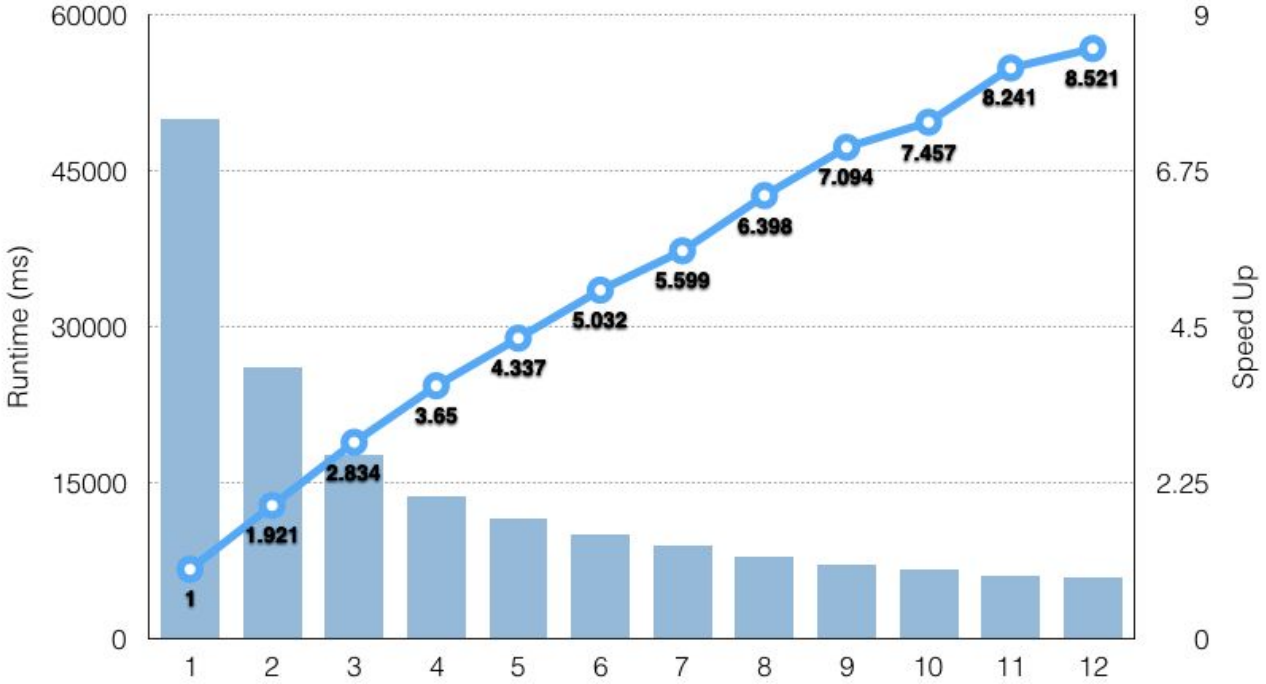


Performance: ExchangeSeqScanExecutor

- 100000 records/tile group * 1000 tile groups
- Less tile groups
  - Less communication & synchronization overhead
  - Less data structure overhead
  - Less result-passing overhead
  - Coarser task granularity, may lead to uneven work division

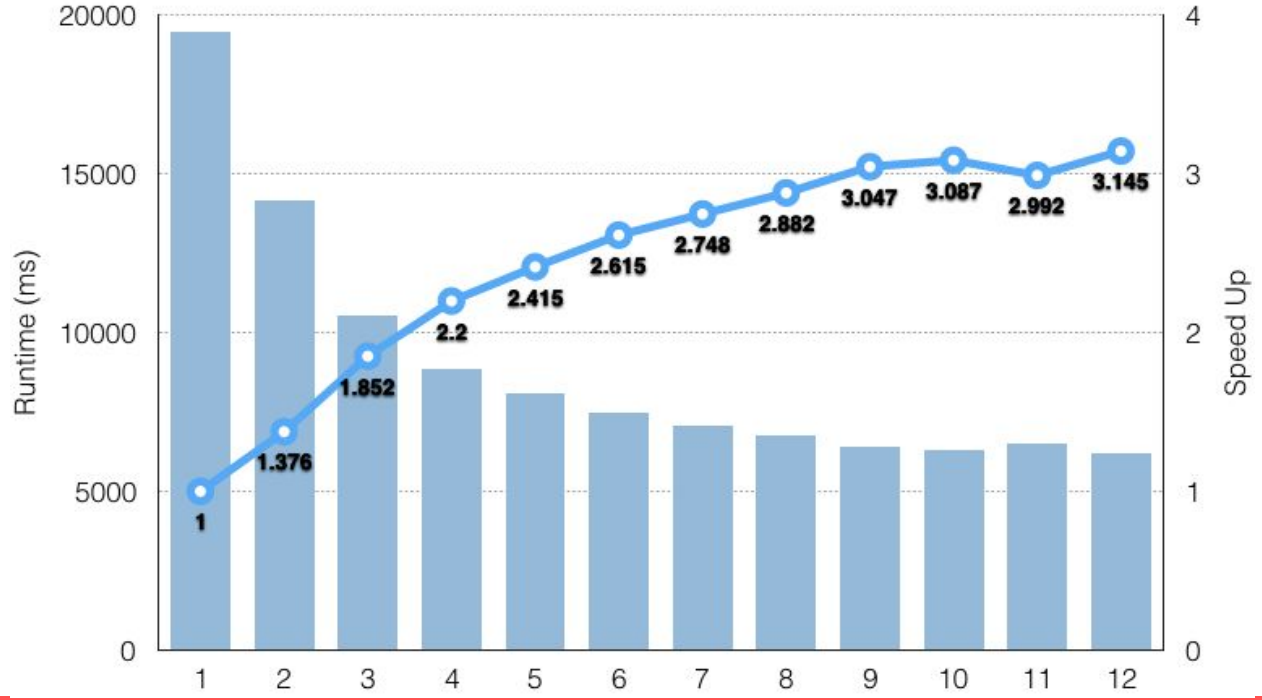
Performance: ExchangeSeqScanExecutor

- 100000 records/tile group * 1000 tile groups



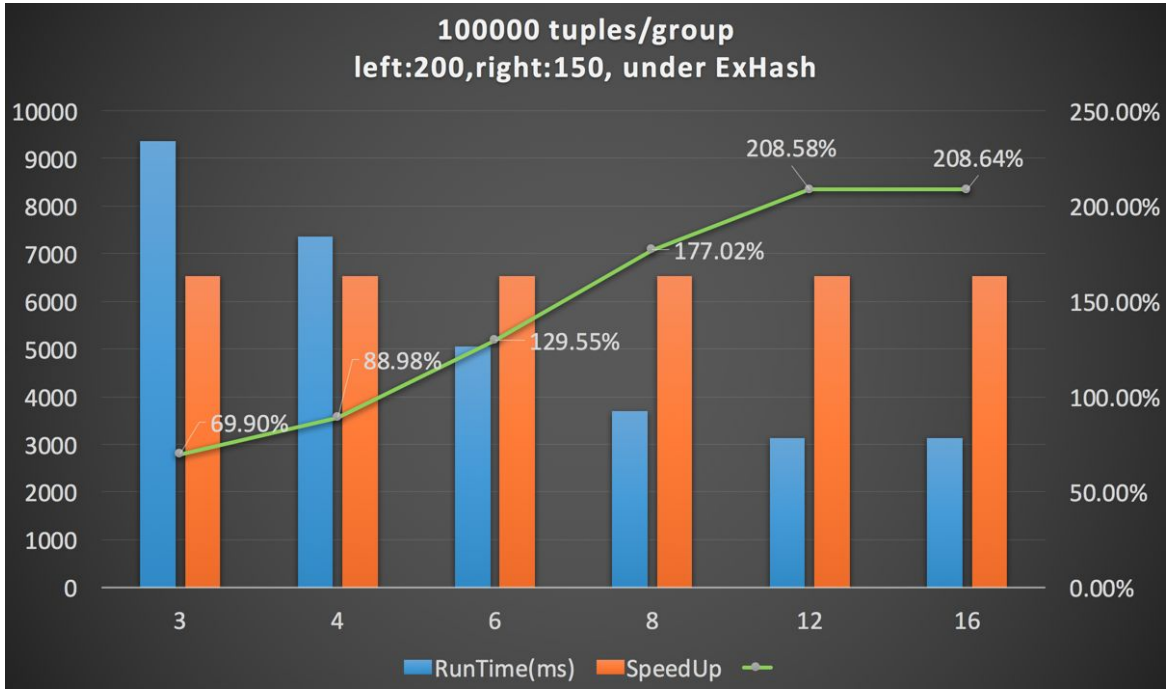
Performance: Hash

- 100000 records/tile group * 300 tile groups



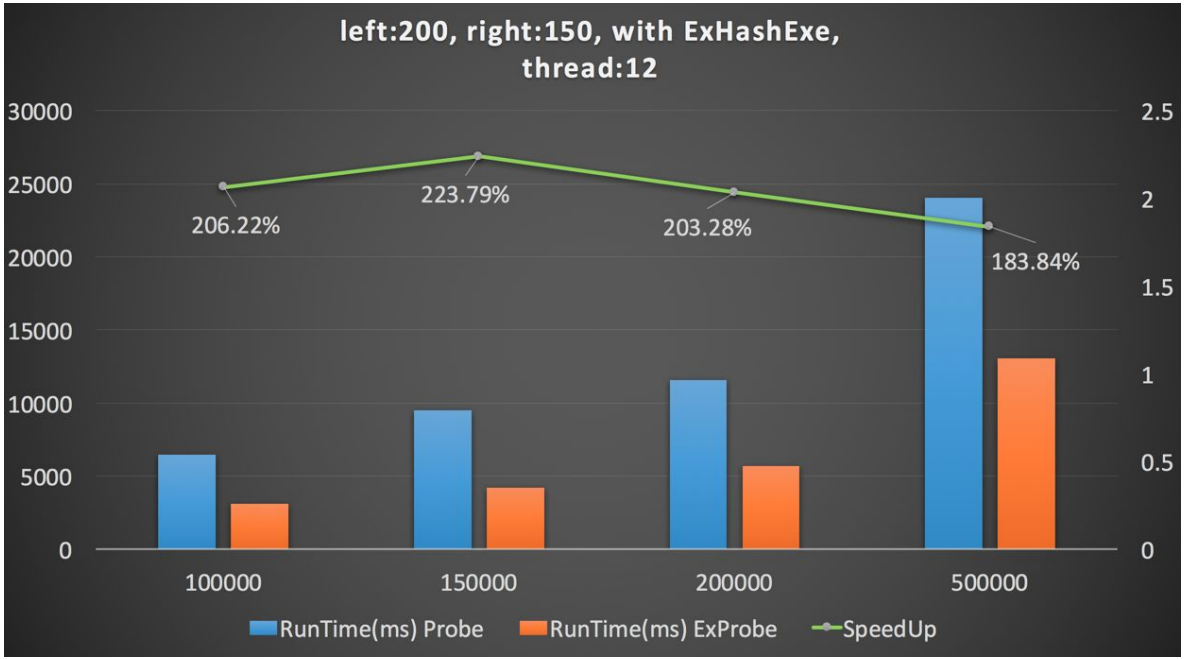
Performance: HashJoin (Probe Phase)

- Experiment Set1: #Thread



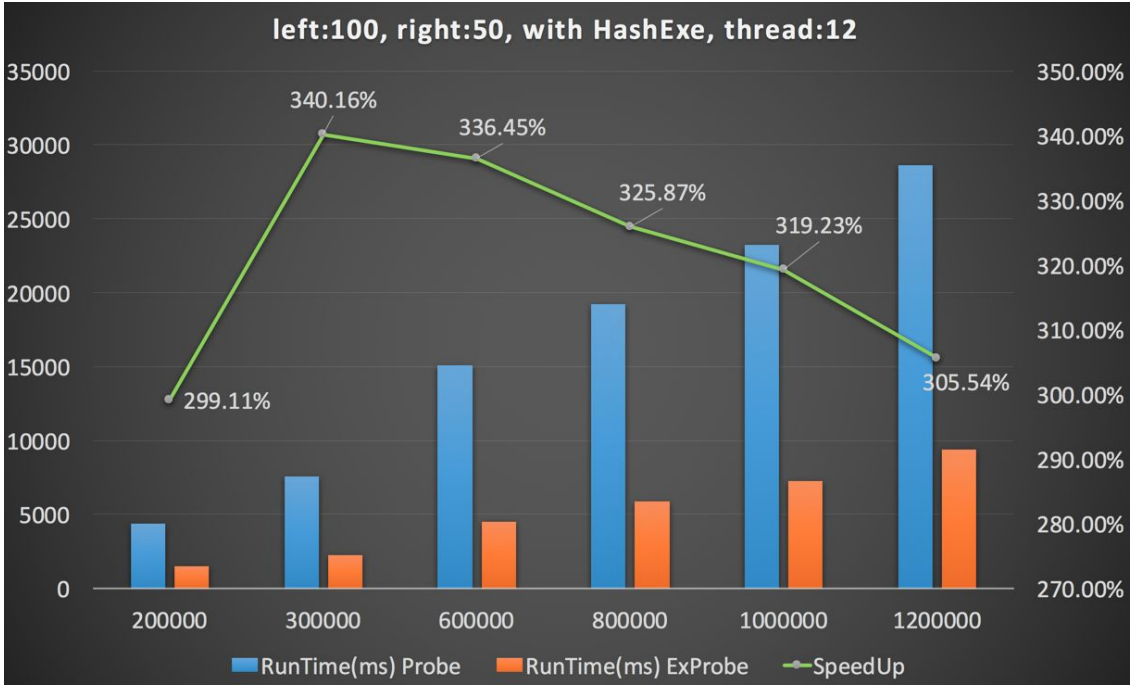
Performance: HashJoin (Probe Phase)

- Experiment Set2: Probe Speedup with ExchangeHashExecutor



Performance: HashJoin (Probe Phase)

- Experiment Set3: Probe Speedup with HashExecutor



Original Goal

Completed

- 75%
  - Ability to insert exchange operators
  - Thread pool and divide work into multiple tasks
- 100% Implement exchange operators
  - Sequential Scan
  - Hash
  - Hash Join

Not Completed

- 100% Implement exchange operators
  - Other executors like aggregate
- 125% NUMA-aware data placement and task scheduling

Future Work

- Improve hash executor and hash join executor (probe phase)
  - May want to reduce contention among threads
- Implement more exchange operators
  - Aggregate Executors
- Create plan nodes for exchange operators
  - If want to have exchange-operator specific data

Thanks