SIGMOD 2025 Programming Contest – Team JobSeeking

Xiangyun Ding   Letong Wang

Overview

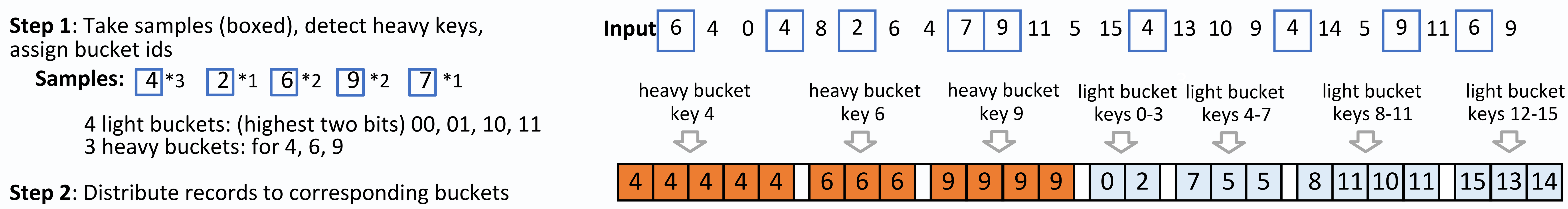
- Our team
  - Team name: **JobSeeking**, from University of California, Riverside
  - Final result: **5'th place**!
  - We presents a high-performance, multi-threaded query engine designed for the SIGMOD 2025 Programming Contest.
- Key techniques
  - A **work-stealing scheduler** for shared-memory multi-core parallelism
  - Highly parallelized I/O modules for rapid data ingestion and emission
  - **Skew-aware parallel hash join** algorithm based on parallel integer-sort
  - Custom memory allocators

Input/Output

- Input Data Format
  - ColumnarTable -> Column -> Page
  - A tree structure for join order
- Parallel Column Read
  - **Parallel prefix sum**: we first perform a parallel scan over pages to sum up the number of rows, which gives each thread the precise starting offset in the array.
  - I/O cache: a column **cache**, keyed by a hash of the column's page content, stores pointers to already-read columns. This cache is crucial for performance.
- Two-Phase Parallel Writing
  - Counting Phase: Determine the number of pages required for each partition of the output.
  - Writing Phase: Allocate memory for pages, then format and write data partitions into the appropriate pages in parallel.

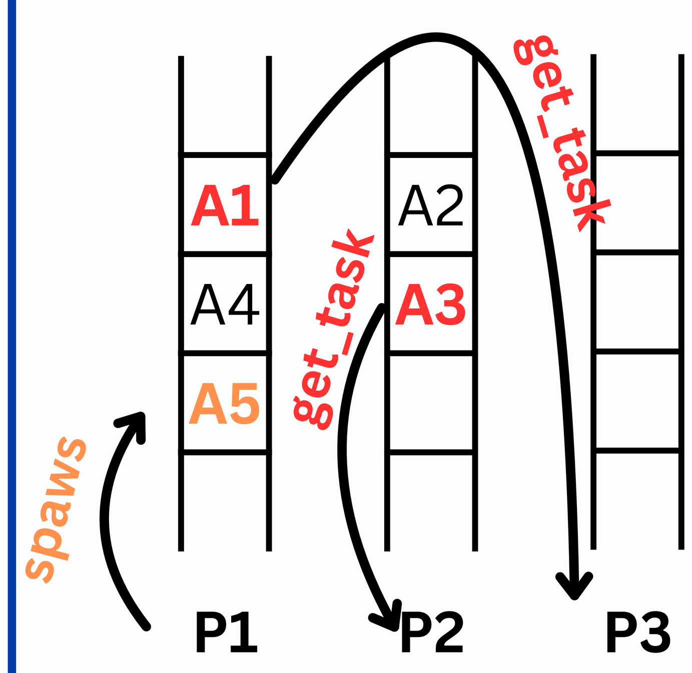
Skew-Aware Parallel Hash Join

- Building the hash table: heavy/light Key partitioning via parallel integer-sort [1]
  - Heavy key identification: by sampling the keys and counting their frequencies, we identify **heavy keys** that appear more often than a dynamic threshold.
  - For heavy keys, we use a hash table to store them with their indices.
  - Light keys are distributed uniformly based on the high bits in parallel. We build one hash table for each light bucket.
  - For queries, we first look up the hash table for heavy keys. If the key is not found, we then look up the corresponding light-bucket hash table.
  - The hash table is implemented by linear-probing.
- Runtime optimizations
  - Dynamic **join order selection**: we greedily choose the pair of tables whose product of row counts is minimized.
  - Dynamic build: the smaller table is always chosen as the build side to construct the hash table.
  - Hash table caching: completed hash tables for large tables can be **cached** and reused in subsequent joins on the same key, avoiding redundant build costs.



Multi-Thread Parallelism

- Work-Steal Scheduler [2]



- Each worker maintains a **deque** for tasks
- Each worker supports:
  - **spaws**: pushes a task to the bottom of deque
  - **get_task**:
    - if the deque is *not empty*, it pops from the *bottom*
    - if the deque is *empty*, it **steals** a task by popping from the *top* of another deque

- Parallel primitives:
  - **parallel-for**
    - Simple application: parallelize any for-loop
    - Adaptive granularity: adaptively adjust task size for high-performance on general machines
  - **scan (prefix sum), interger sort, ...**
- Parallel resources allocator
  - Allocate **thread pool** with size #cores/#cores*2
  - Allocate **memory pool** for intermediate results of parallel functions

References

[1] Dong, Xiaojun, Laxman Dhulipala, Yan Gu, and Yihan Sun. "Parallel integer sort: Theory and practice." In Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, pp. 301-315. 2024.
[2] Blelloch, Guy E., Daniel Anderson, and Laxman Dhulipala. "ParlayLib-a toolkit for parallel algorithms on shared-memory multicore machines." In Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures, pp. 507-509. 2020.