Cache-Efficient Aggregation: Hashing Is Sorting

Ingo Müller, Peter Sanders, Arnaud Lacurie, Wolfgang Lehner, Franz Färber
SIGMOD, June 3, 2015
Hash-Aggregation
- Insert every row into hash map with grouping attributes as key
- Aggregate to existing intermediate result

Sort-Aggregation
- Sort input by grouping attributes
- Aggregate consecutive rows in a single pass

Traditional approach
- Optimizer selects physical operator based on cardinality estimation → error prone.

Our goal: Hashing and Sorting in a single operator.
Mixing Hashing and Sorting (1/3): Idea

Key observation: Hashing is the same as Sorting by hash value.

General idea:
• design an aggregation operator like a Divide’n’Conquer sort algorithm on the hash values of the grouping attributes.

Common technique:
• combine different sort routines into one algorithm.
Mixing Hashing and Sorting (2/3): Example Execution

input: (hash, group, value)

1st level of recursion

Hashing

Partitioning

Partitions:

2nd level of recursion

result:
Mixing Hashing and Sorting (3/3): Recap

**Our approach**: aggregation algorithm designed like a *sort algorithm on hash values* with built-in aggregation.

Subroutine “Hashing”:
- Inserts into a series of hash tables (like insertion sort)
- Each of cache size → efficient (sort of)
- Does the actual aggregation

Subroutine “Partitioning”:
- Appends to hash partitions (like radix sort)
- Only sequential access → efficient
- Does no aggregation

Next question: when to use which routine?
"HashingOnly": in cache for small output size, slow recursive processing otherwise

"PartitionAlways":
- Much faster partitioning (97% of speed of `memcpy` thanks to “Radix-Partitioning”)
- No (early) aggregation → induced useless work for small output

Goal: use Hashing iff working set fits into cache.
Adaptation Mechanism (2/2)

Adaptive algorithm:

- **Partitioning** recurses when necessary
- **Hashing** ends recursion when possible efficiently

Our mechanism finds the right strategy *adaptively.*
Evaluation: Comparison with Prior Work

State of the art:
- Implementations of:
- 1-pass algorithms:
  - Hybrid
  - Atomic
- 2-pass algorithms:
  - Partition and Aggregate
  - Independent
  - PLAT

Result:
- “Adaptive” faster for $K > 2^{20}$
- Up to factor 3.7 speedup

Recursive processing is crucial for large outputs.

Summary and Outlook

- Observation: Hashing is **Sorting by hash value**.
- We can **combine them in a single algorithm** to combine their advantages.
- **Adaptation mechanism** provides robust, optimal performance up to factor 3.7 faster than prior work.

- What else to expect in the paper:
  - How to **parallelize**? How to integrate with **JiT** and **column-wise processing**?
  - How to tune hashing and sorting to **modern hardware**?
  - How to determine **thresholds**?
  - Why does it also work well in presence of **skew**?

Thank you