Efficient Sorting, Duplicate Removal, Grouping, and Aggregation (2022)

By: Thanh Do, Goetz Graefe, Jeffrey Naughton

Presented by Ian Gatlin

What is this paper about?

Database query processing requires algorithms for duplicate removal, grouping, and aggregation.

In SQL you can chain operations together in a single query. Based on earlier operations, later operations may be faster based on groupings/sorts from the output of the previous operation.

This paper focuses on an algorithm that performs well on run-time efficiency in both cases.

Applications of these ideas

SQL duplicate removal are used in applications of all kinds:

- The example given in the paper is removing duplicate users from a website log. This can reduce billions of rows to millions of rows.
- In my UROP, different parts of the genome may have same effects on the upstream and downstream regulators

Duplicate Removal

- SELECT DISTINCT Color FROM Foods
- Returns unique values

Food	Color
Strawberry	Red
Spinach	Green
Raspberry	Red
Lettuce	Green

$\longrightarrow$

Color
Red
Green

Grouping and Aggregation

- SELECT Color, Count(*) FROM Foods GROUP BY Color
- Creates group objects where data is sorted into contiguous chunks, and then applies aggregation function that calculates value based off group

Food	**Color**
Strawberry	Red
Spinach	Green
Raspberry	Red
Lettuce	Green

$\longrightarrow$

Color	**Count**
Red	2
Green	2

Existing Implementations

- In-stream aggregation - This is the most efficient implementation, but it requires sorted input

If the input is not sorted:

- Sort-based aggregation - uses an external memory merge sort. Relatively slower and uses more memory, but good if sorted order can help with subsequent operations
- Hash-based aggregation - uses in-memory hash table + hash partitioning to temporary storage. Aggregates with hash buckets, this is good when the hash table (the output of the aggregation), can fit into memory.
- Uses estimation in compile time to determine whether to use sort or hash based aggregation in this step

What is the problem with this?

- The estimations for input and output size isn’t reliable and is often inaccurate during query compilation
- Because of this, the wrong algorithm choice is often made, causing unpredictable performance and slower runtimes
- Having unpredictable performance allows developers to better account for performance delays, which can help make better user interfaces

Contribution of Paper

- This paper aims to create a new in-sort aggregation algorithm to replace the sort-based and hash-based aggregation algorithms with a single algorithm.
- The goal is to make it competitive with the hash based algorithm, while creating a sorted order that will help with previous runs.
- Now, instead of having to rely on erroneous estimates, database engines can rely on a single algorithm

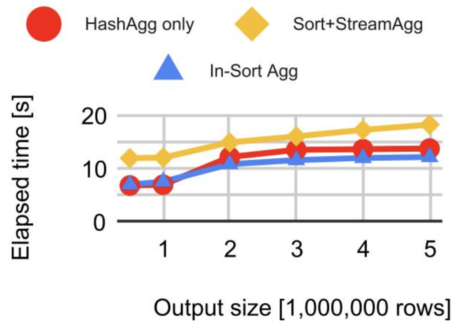
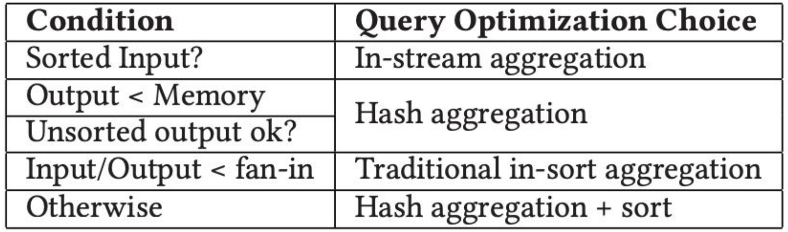


Fig. 3. Motivating performance example from Google's F1 Query.

Benefits

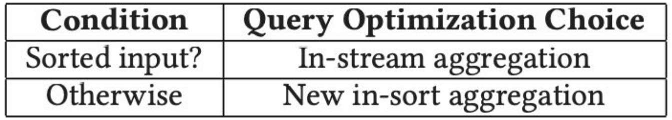
- Less production code means less maintenance and more readability and understandability
- Reduces unpredictable performance
- Easier to understand how to optimize queries
- Easier to analyze runtime and bottlenecks in specific queries

Table 1. Traditional Decision Procedure



Condition	Query Optimization Choice
Sorted Input?	In-stream aggregation
Output < Memory	Hash aggregation
Unsorted output ok?	
Input/Output < fan-in	Traditional in-sort aggregation
Otherwise	Hash aggregation + sort

Table 2. Decision Procedure with the New Algorithm



Condition	Query Optimization Choice
Sorted input?	In-stream aggregation
Otherwise	New in-sort aggregation

How can we optimize these operations?

- We can’t change the input or output size of the data, but we can control how much temporary storage we use in the grouping and aggregation of data to achieve the output.
- This is what the paper focuses on optimizing

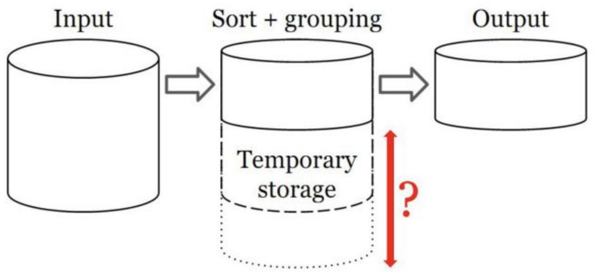


Fig. 1. Optimization opportunity in sorting and grouping.

Introducing the Implementation

- This paper introduces two new techniques. They give sorting the property of early aggregation by using in memory binary tree indexes, and using wide merging to reduce the amount of temporary storage needed.
- Reducing memory is the name of the game.

Technique 1: Early Aggregation

- It allows for aggregation in sorting data structures by using a binary tree (rather than priority queues) that can eliminate duplicate values in the primary runs to temporary storage (during sorting) rather than after the merge. This uses less temporary memory.

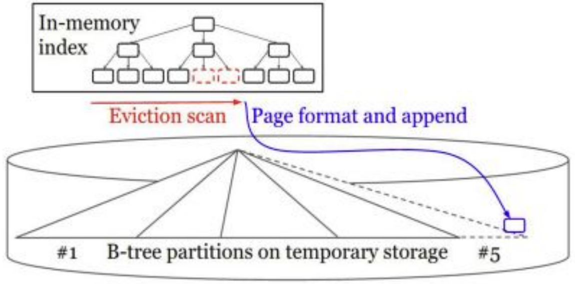


Fig. 6. Run generation using an ordered in-memory index.

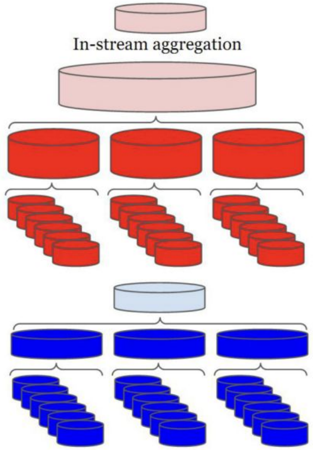


Fig. 2. In-stream duplicate removal after sort (red) vs. in-sort duplicate removal within runs (blue).

Technique 2: Wide Merging

- The second new technique, wide merging, improves the final merge step of external merge sort by having a larger fan-in.
- Rather than using a memory page for each input, wide merging uses a single page for all inputs. It absorbs inputs from others into a single page.
- Reduces the amount of memory necessary for merge.

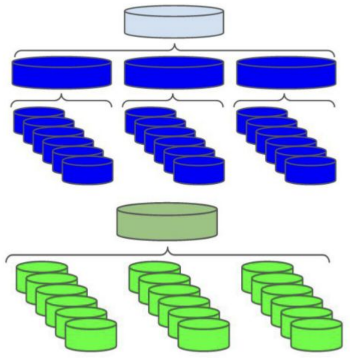


Fig. 8. The effect of wide merging (green).

Performance - CPU Time

- CPU Time - Speed of queries

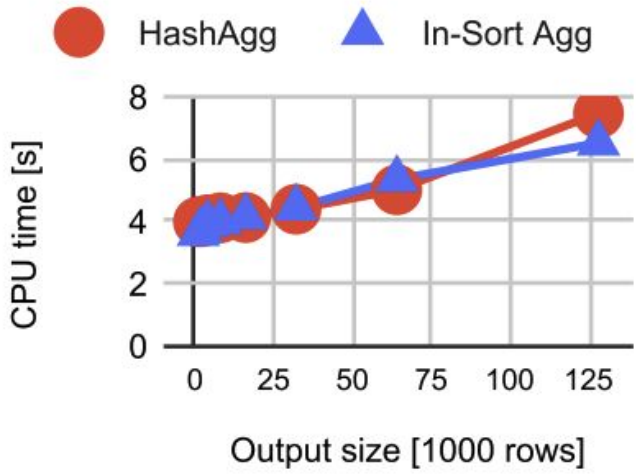


Fig. 15. Performance of in-memory indexes.

Performance - Spillage

- Spillage - amount of external temporary storage used. We want to reduce the blow-up factor of this if output is greater than the size of memory. Sometimes it's unavoidable

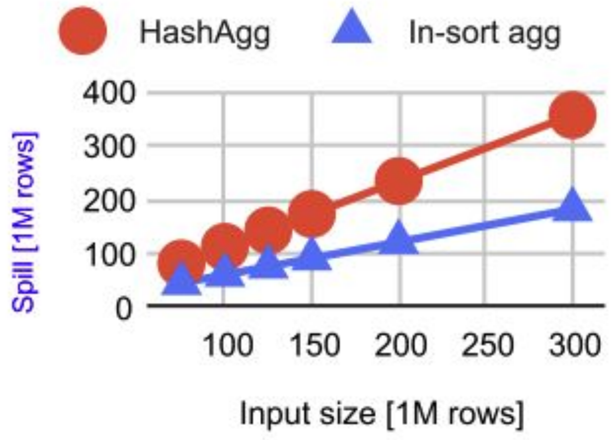


Fig. 18. Spill volume.

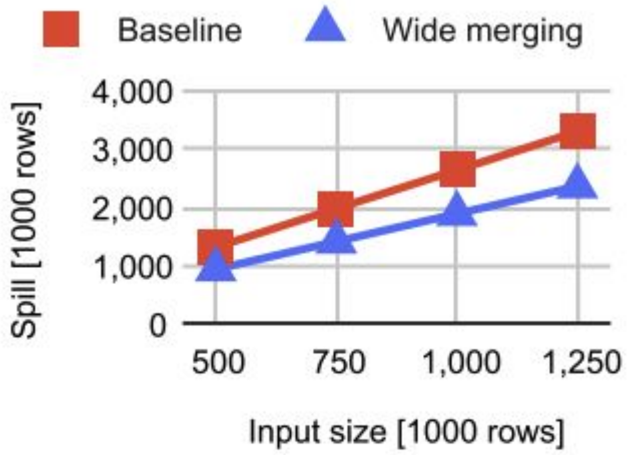


Fig. 14. Effect of wide merging.

Conclusions

- This paper introduces a new that is competitive with the hash based algorithm, while creating a sorted order that will help with subsequent runs.
- Database engines can rely on a single more reliable algorithm
- Less production code means less maintenance and more readability, and understandability

Pros

- Very thorough paper with many theoretical and experimental justifications
- Competitive with both previous approaches with sorting benefit

Cons

- More figures to explain how it works would be useful
- Some number dense paragraphs in paper are not as readable

Thank you! Questions