In-memory Aggregation for Big Data Analytics

UNB EST. 1785 UNIVERSITY OF NEW BRUNSWICK
Computer Science

Puya Memarzia, Virendra C. Bhavsar, and Suprio Ray

Faculty of Computer Science, University of New Brunswick, Fredericton, New Brunswick, Canada

ABSTRACT

Aggregation is widely used to extract useful information from large volumes of data. In-memory databases are rising in popularity due to the demands of big data analytics applications. Many different algorithms and data structures can be used for in-memory aggregation, but their relative performance characteristics are inadequately studied. Prior studies in aggregation primarily focused on small selections of query workloads and data structures. We undertook a comprehensive analysis of in-memory aggregation that encompasses 20 popular and state-of-the-art algorithms and data structures. Insights gained from theoretical and empirical evaluation are used to identify the trade-offs of each algorithm, with the goal of offering insights to practitioners. Our results allowed us to identify the best approach in different situations, based on specific characteristics of the query workload and dataset.

Motivation

- Aggregation: a ubiquitous and expensive operation commonly used in data analytics
  - Applications: data warehousing, data mining, business intelligence tools, …
- Plethora of algorithms and data structures could be used to implement aggregation
- What are the tradeoffs and opportunities to improve performance?
- How to select the best tool for the job?

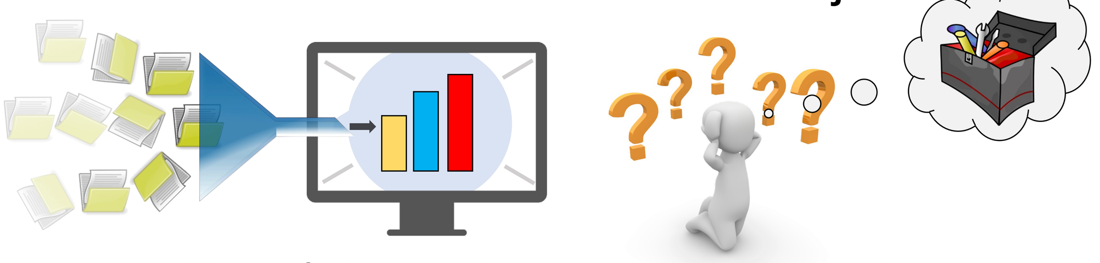


Figure 1. Aggregation overview and motivation

In-memory Query Processing

- Memory getting faster, denser, and cheaper
- Significant speedups (orders of magnitude) over disk-based query processing
- Data resides in main memory
- Performance highly influenced by memory access patterns [1]

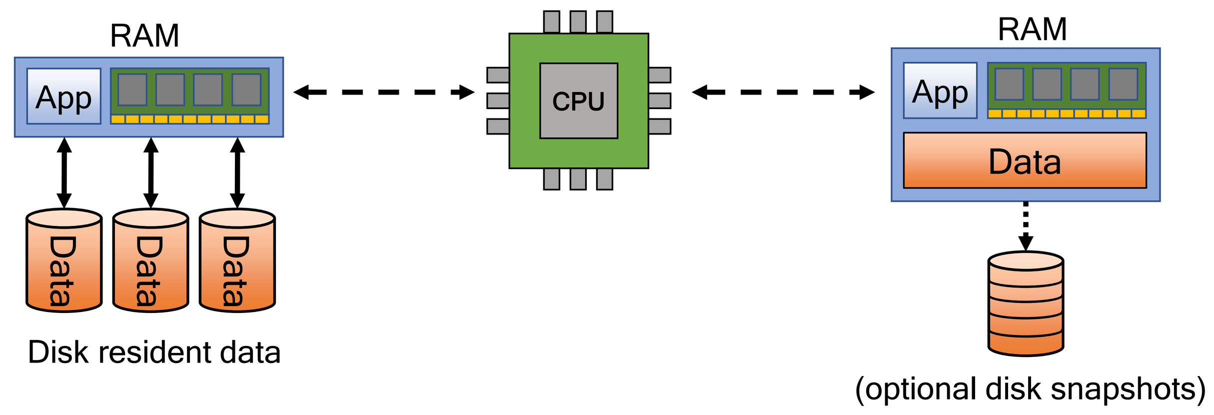
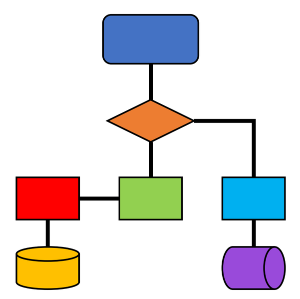


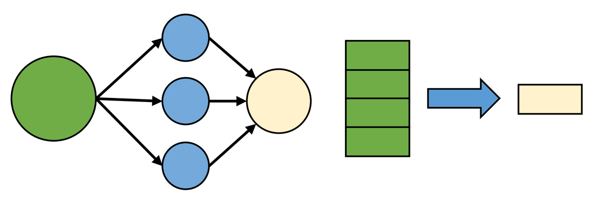
Figure 2. Comparison of disk-based and memory-based approaches

1. Algorithm and Data Structure



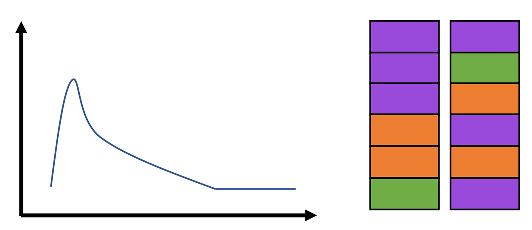
- Hash-based, sort-based, and tree-based approaches
- Including popular, state-of-the-art, and custom implementations

2. Query and Aggregate Function



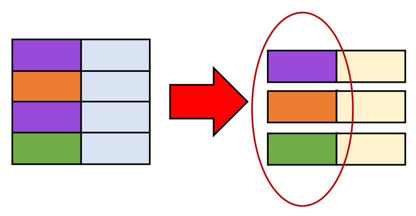
- Aggregation query categories
  - Scalar or Vector
  - Distributive, Algebraic, or Holistic
  - Range predicate

3. Key Distribution and Skew



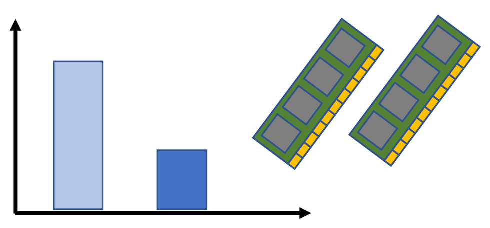
- Data key distribution (Zipfian, heavy hitter, moving cluster, etc.)
- Data ordering (randomness/locality)

4. Group-By Cardinality



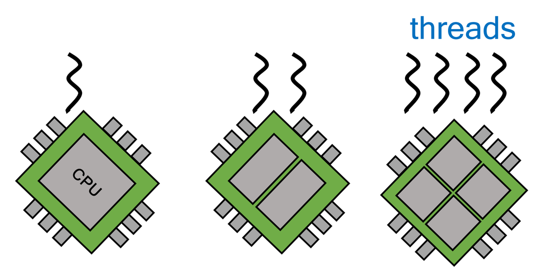
- Number of unique values in group-by columns
- Determines size of output

5. Dataset Size and Memory Usage



- Size of input data
- Algorithm memory efficiency
- Query memory requirements

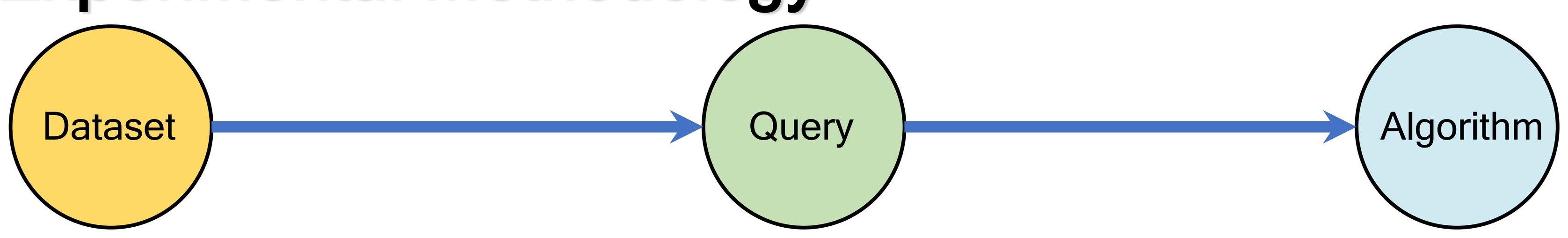
6. Concurrency and Multithreaded Scaling



- Support for concurrent (shared memory) access
- Ability to scale up with additional threads

Figure 3. Six analysis dimensions (important factors that can be evaluated independantly)

Experimental Methodology



- Six dataset distributions
- Extends popular datasets originally proposed in [2]
- Up to 100M key-value pairs
- Variable cardinality/skew

- Seven query workloads
- Covers all fundamental aggregation categories

- Initial pool of 20 algorithms selected for evaluation
- Microbenchmarks used to filter out inefficient implementations

Distributive query example (Q1)

```
SELECT product_id, COUNT(*)
FROM sales GROUP BY product_id
```

Holistic query example (Q3)

```
SELECT product_id, MEDIAN(amount)
FROM sales GROUP BY product_id
```

Table 1. Algorithms and Data Structures

Label	Type
ART	Tree
Judy	Tree
Btree	Tree
Hash_SC	Hash Table
Hash_LP	Hash Table
Hash_Sparse	Hash Table
Hash_Dense	Hash Table
Hash_LC	Hash Table
Hash_TBBSC	Hash Table
Hash_LCMC	Hash Table
Introsort	Sort Algorithm
Spreadsort	Sort Algorithm
Sort_QSLB	Sort Algorithm
Sort_BI	Sort Algorithm

Experimental Results

- Evaluating the performance impact of each of the six dimensions
- Measured runtime with accurate timer, reporting average of five runs
- Please see paper [3] for additional results and details

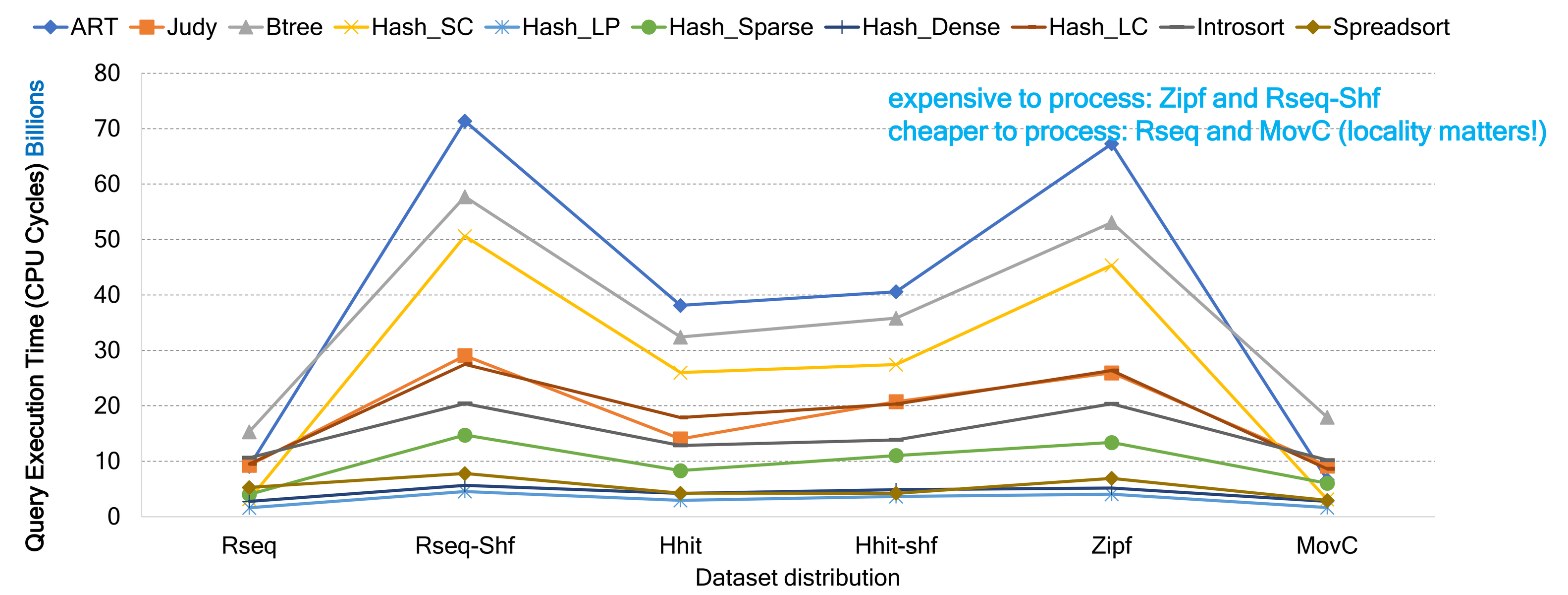


Figure 4. Variable dataset distribution - Vector COUNT (Q1) - 100M records - 1M group-by cardinality

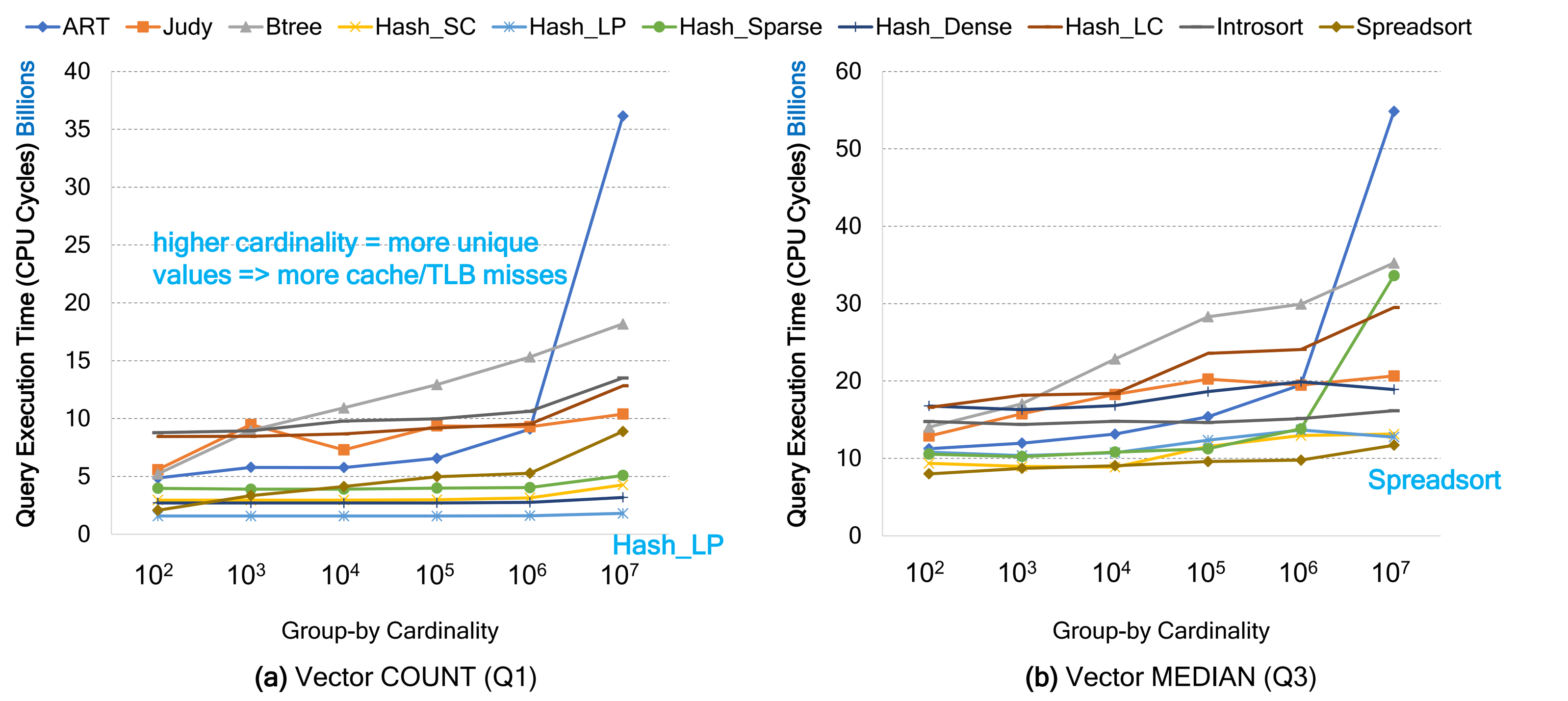


Figure 5. Variable cardinality - 100M records

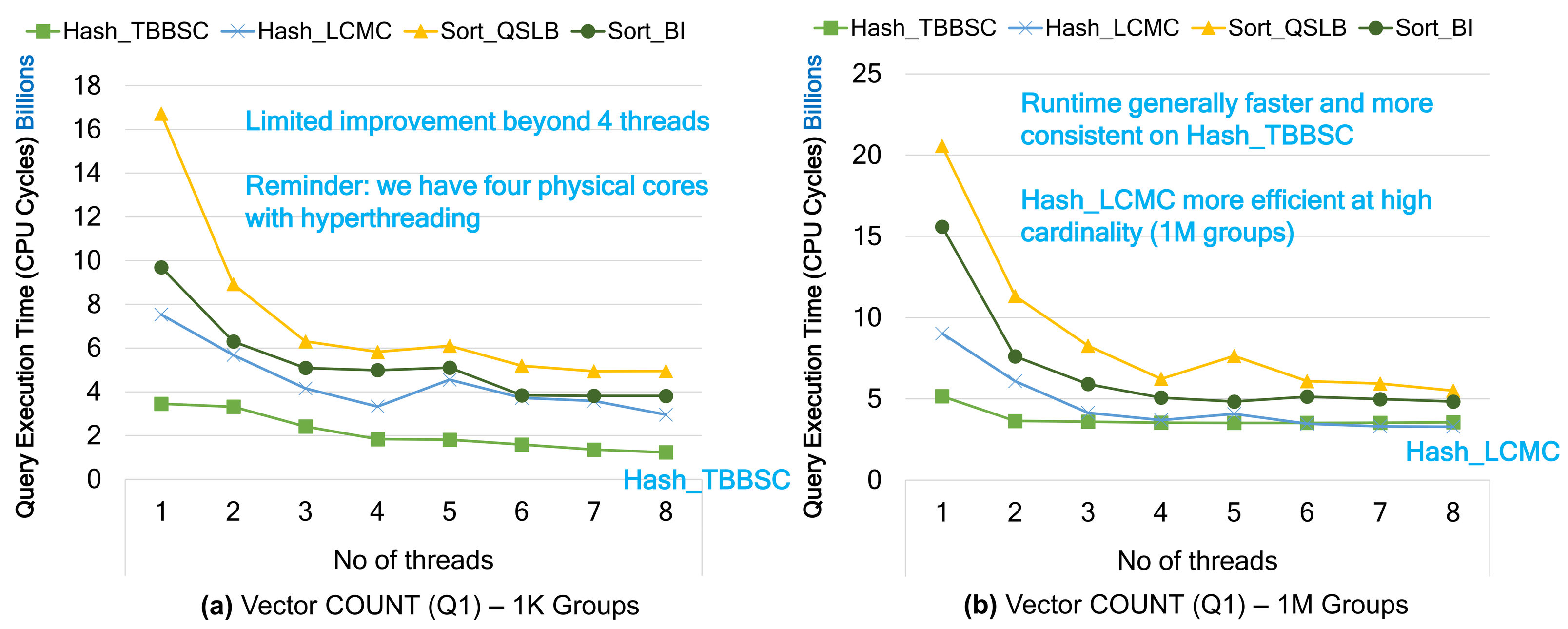


Figure 6. Multithreaded scaling (concurrent algorithms) - 100M records

Conclusions

- Aggregation heavily impacted by data and workload characteristics
- Sort-based approaches: generally perform best on holistic workloads
- Hash-based approaches: generally the fastest on distributive workloads
- Tree-based approaches: too slow for write-once-read-once (WORO) workloads. Potential for write-once-ready-many (WORM) workloads or situations requiring dynamic growth

References: [1] Puya Memarzia, Suprio Ray, and Virendra C. Bhavsar, "On Improving Data Skew Resilience In Main-memory Hash Joins", 22nd International Database Engineering & Applications Symposium (IDEAS), May 2018 [2] Gray, Jim, Prakash Sundaresan, Susanne Englert, Ken Baclawski, and Peter J. Weinberger. "Quickly generating billion-record synthetic databases." In Acm Sigmod Record, vol. 23, no. 2, pp. 243-252. ACM, 1994. [3] Puya Memarzia, Suprio Ray, and Virendra C. Bhavsar, "A Six-dimensional Analysis of In-memory Aggregation", International Conference on Extending Database Technology (EDBT-2019), 2018.

Sponsored by: NBIF NSERC CRSNG